TRANSFORMATION OF THE UNIVERSITY EXAMINATION TIMETABLING PROBLEM SPACE THROUGH DATA PRE-PROCESSING

SITI KHATIJAH NOR ABDUL RAHIM, BSc., MSc.

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

2015

Abstract

This research investigates Examination Timetabling or Scheduling, with the aim of producing good quality, feasible timetables that satisfy hard constraints and various soft constraints. A novel approach to scheduling, that of transformation of the problem space, has been developed and evaluated for its effectiveness.

The examination scheduling problem involves many constraints due to many relationships between students and exams, making it complex and expensive in terms of time and resources. Despite the extensive research in this area, it has been observed that most of the published methods do not produce good quality timetables consistently due to the utilisation of random-search. In this research we have avoided random-search and instead have proposed a systematic, deterministic approach to solving the examination scheduling problem. We pre-process data and constraints to generate more meaningful aggregated data constructs with better expressive power that minimise the need for cross-referencing original student and exam data at a later stage. Using such aggregated data and custom-designed mechanisms, the timetable construction is done systematically, while assuring its feasibility. Later, the timetable is optimized to improve the quality, focusing on maximizing the gap between consecutive exams. Our solution is always reproducible and displays a deterministic optimization pattern on all benchmark datasets. Transformation of the problem space into new aggregated data constructs through pre-processing represents the key novel contribution of this research.

Publications / Disseminations during PhD period

Papers Published / Presented:

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2009. Granular Modelling Of Exam To Slot Allocation. ECMS 2009 Proceedings edited by J. Otamendi, A. Bargiela, J. L. Montes, L. M. Doncel Pedrera (pp. 861-866). European Council for Modelling and Simulation. (doi:10.7148/2009-0861-0866).

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2012. Deterministic Optimization of Examination Timetables. In 25th European Conference on Operational Research, EURO 2012, Session WC-14, p.220-221, Vilnius, Lithuania, July 2012.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2013. Hill Climbing Versus Genetic Algorithm Optimization in Solving the Examination Timetabling Problem. 2nd International. Conference On Operations Research and Enterprise Systems, ICORES 2013, Barcelona, Spain, 16-18 February 2013.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2013. Domain Transformation Approach to Deterministic Optimization of Examination Timetables, Artificial Intelligence Research (AIR) Journal. Sciedu Press. 2(1), 2013. (doi:10.5430/air.v2n1p122).

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2013. Analysis of Backtracking in University Examination Scheduling. In proceeding of: 27th European Conference on Modelling and Simulation, ECMS2013, At Aalesund, Norway. doi: (10.7148/2013-0782)

Rahim, S. K. N. A., Bargiela, A., & Qu, R. 2013. A Study on the Effectiveness of Genetic Algorithm and Identifying the Best Parameters Range for Slots Swapping in the Examination Scheduling. International Symposium on Mathematical Sciences and Computing Research, iSMSC 2013. Ipoh, Malaysia.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. The Incorporation of Late Acceptance Hill Climbing Strategy in the Deterministic Optimization of Examination Scheduling Framework: A Comparison with the Traditional Hill Climbing. 2014 IEEE Conference on Systems, Process and Control (ICSPC 2014), 12 - 14 December 2014, Kuala Lumpur, Malaysia.

Acknowledgement

I would like to take this opportunity to personally thank a number of people for their help and support during my PhD study. First and foremost, I would like to express my profound gratefulness and deep honours to my first supervisor, Professor Andrzej Bargiela for his constant guidance, support, help, encouragement, and constructive comments throughout this study. I really appreciate his dedicated commitments and professionalism in supervising my research study.

Also, I would like to express my gratitude to Dr Rong Qu, my second supervisor, for all the positive, constructive and detailed comments in my work that she has always given to me despite her busy schedule and commitments.

I would also like to thank the Ministry of Higher Education (MOHE) of Malaysia and the Universiti Teknologi MARA (UiTM) for the doctoral scholarship and other financial support throughout the course of this study. My thanks also go to my wonderful friends and staffs at the University of Nottingham Malaysia Campus and University Teknologi MARA (UiTM) for their advice, support, and friendship.

A special thanks to my lovely husband Amir Hamzah Jaafar for his great support, patience, understanding and unconditional love. Finally, thank you very much to my both mother and mother-in-law, my daughters, my brother and sisters, the entire family and friends for their support, love and prayers.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

There are many events and activities in this world that need to be synchronized. From social community activities, work and transportation to personal agendas, they all need to be planned and scheduled. The effectiveness of all this planning depends on the efficiency of the schedules. This thesis is focused on transforming the university examinations' scheduling problem into a more structured domain, in which a new representation of information through pre-processing is introduced. We also studied and implemented a few optimization approaches that enhance the solutions generated with the proposed approach.

This chapter presents the introduction to this research, followed by the scope and objectives of this study. Later, we present the thesis contributions in brief. Finally the thesis overview is specified which briefly explains how this thesis is organized, chapter by chapter.

1.1 Introduction

The word "timetable" (also known as schedule) is defined by the Oxford Advanced Learner's Dictionary (which can be accessed from http://www.oxfordlearnersdictionaries.com) as *"a list showing the times at which particular events will happen"*. Therefore, timetabling or scheduling

can be thought of as a process of creating schedules that will list events and the times at which they are planned to occur. In many organizations or institutions, scheduling is an important challenge and is considered a very tedious and time-consuming task. Normally, the personnel involved in preparing the schedules will do it manually and, in most cases, using a trial-and-error approach. Some scheduling problems involve many constraints, and due to this the preparation of the schedules sometimes becomes complex and expensive in terms of time and resources.

Wren (1996) mentioned that timetabling and scheduling has a special type of relationship. The author defined timetabling as follows:

"Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives."

There are various areas of scheduling, which include educational scheduling, sports scheduling, transportation scheduling and nurse scheduling, etc. Due to the wide spectrum of applications of scheduling, research in the area is also scattered and is usually problem-specific. Scheduling research not only concentrates on generating a feasible timetable but the efficiency of the solution generated is also sought after. Numerous approaches or methods have been proposed since the 1960s by researchers from the Operational Research and Artificial Intelligence area, as surveyed by Qu *et al*. (2009a).

Among the broad areas of the scheduling problems, educational scheduling is one of the most studied and researched areas in the scheduling literature. This is due to the significant and time-critical

challenge associated with the requirement of preparing the schedules periodically in schools, colleges and universities (quarterly, annually etc.).

Educational scheduling includes school scheduling (course/teacher scheduling), university course scheduling, university examination scheduling and more. For this scheduling problem, in most universities nowadays, the students are given the flexibility to enrol for courses across faculties. That makes this kind of scheduling problem more challenging and expensive to solve. In some cases, a number of people are in charge of producing the schedules, and thousands of hours have been spent on this.

As an example, Universiti Teknologi Mara (UiTM) which is Malaysia's largest institution of higher learning in terms of size and population is no different in generating schedules. Besides the main campus in Shah Alam, UiTM has expanded nationwide with 12 state campuses, 6 satellite campuses in Shah Alam, 11 state satellite campuses and 21 affiliated colleges (http://www.uitm.edu.my/index.php/en/about-uitm/uitm-profile-history/university-profile). This university offers more than 500 academic programmes delivered by 24 faculties. The schedules will be prepared each semester by the timetable committee which exist in every faculty. The committee is responsible to come up with a complete schedule, which relates the lecturers, student groups and rooms. Unlike other universities, UiTM has a different policy in disseminating information to the students, most lectures are being conducted in small classes with a minimum of 15 and a maximum of 40 students, which introduces additional constrains to the preparation of the schedules.

In a different perspective, we have examined the number of resources utilized to generate the schedules each semester. For a typical UiTM branch campus having 25 departments, each department will have a minimum of two persons as a committee member, with a total of 50 persons involved in the whole exercise which constitute roughly about 16% of the total faculty members. In preparing the course schedules, 40 working hours will be required by each committee member, in overall the whole exercise consumes 2000 hours. The time spent on producing schedules in a large educational establishment may not be obvious; however, cumulatively and collectively it is equivalent to the time that may be spent to build an airplane (Wilson R, 2010).

Surveys and overviews of educational timetabling problems and the proposed methods to solve them can be found in many publications e.g. (Schmidt and Strohlein, 1980), (Carter, 1986), (Carter and Laporte, 1996), (Burke *et al.,* 1997), (Schaerf, 1999), (Qu *et al.,* 2009a), (Pillay, 2013), (Kristiansen and Stidsen, 2013) and etc.

In this work, the focus is the university examination scheduling problem. This problem is known as an NP hard real world problem (Cooper and Kingston, 1996; and Even *et al.*, 1976). This problem has increasingly become more challenging in recent years due to the raise in students' enrolments and especially when students are given the flexibility to register modular courses across faculties (Burke *et al.*, 1994a) and (McCollum, 2007).

The standard objective of university examination scheduling problem is to satisfy the most important hard constraint that is to produce

feasible examination schedules (i.e. no conflicting exams scheduled concurrently). However, it is also important to produce a good quality schedules according to some preferences, which can be considered as soft constraints. The term 'soft' refers to the fact that the satisfaction of these types of constraints is not really crucial but the fulfilment will benefit some entities.

To date, the number of approaches or methods proposed to solve examination scheduling problems is increasing. These research efforts have evaluated various approaches, created new methods and produced promising findings or results. Efforts have also been devoted to automating the scheduling process, so that the generation of schedules could be carried out using computer software. However, due to the inherent complexity of the problem, there is still room for improvement in the current state of the art.

Common approaches developed in solving the timetabling problems usually consist of two phases, i.e. the *construction* and *improvement* phase (as claimed by (Hertz, 1991)). With regard to the *constructive* approach, Burke *et al.*, (2010b) stated that a constructive approach begins with an empty solution and additionally constructs a final (complete) solution by utilizing some heuristics. As opposed to the *constructive* phase, the *improvement* phase begins with a complete solution where by the quality of the solution is enhanced (normally using certain procedures repeatedly until the optimal solution is produced).

One of the most widely used method in the construction phase is the graph colouring heuristics, where it is defined as the problem of

colouring vertices of a graph with the most minimum number of colours so that no two adjacent vertices share the same colour. Examination timetabling problem can be represented as a graph colouring problem, where the vertices represent the exams, edges represent the clashes between exams and colours represent the time slots (Carter, 1986), (Broder, 1964), (Cole, 1964), (Peck and Williams, 1966), (Welsh and Powell, 1967), (Laporte and Desroches,1984), (Burke *et al.,* 1994c), (Carter *et al.,* 1994), (Burke and Newall, 2004a), (Asmuni *et al.*, 2009), (Abdul-Rahman *et al.*, 2009), (Kahar and Kendall, 2010) and etc. Therefore, by representing the examination scheduling problem using a graph colouring problem, the main objective is to find the minimum number of time slots to schedule all the exams without any conflicts.

Though graph colouring heuristic is naturally quite simple, however an initial solution with good quality is often produced. Coupled with an improvement phase, many good quality examinations schedules are being produced by the researchers (Carter, 1986), (Carter *et al.*, 1994), (Joslin and Clements, 1999), (Burke and Newall, 2004a), (Asmuni *et al.*, 2007), (Abdul-Rahman *et al.*, 2009), (Kahar and Kendall, 2010) and etc. But despite this fact, the timetabling researchers are aware that there is no single heuristic that can be used to solve all timetabling problems because of the incorporation of problem-specific features in the heuristics. Due to this, current area of research concern is to investigate how to raise the level of generality of state of the art algorithm, in order to deal with a broader range of problems.

The other well known objective of examination scheduling in the literature is to produce good quality timetable, where each exam taken by

individual student should be scheduled as far apart as possible from one another. Carter's evaluation function, proposed by Carter *et al.* (1996) is extensively used by researchers in the literature to measure the quality of examination schedules based on the above mentioned criteria.

1.2 Scope and Objective

In this research, as mentioned above, our focus is the university examination scheduling. As such, besides aiming to propose a method that could generate feasible examination schedules (which is by satisfying the hard constraint, i.e. no conflicting exams are scheduled in the same time slot), we are aiming to improve the quality of the initial examination schedules constructed.

Despite the frequent generation of these schedules which occurs periodically in all universities across the world, we can still see some students having an unfavourable examination schedules. Examples of unfavourable schedules include those where students have two or more examinations in a row. We intend to research into how to improve the existing methods available in solving this problem to ensure that better quality schedules are generated.

To be specific, our main objective is to propose a transformation of the complex university examination timetabling problem space into a more structured domain, in which a new representation of information through pre-processing is introduced. Other objectives are:

- To propose a method (construction phase) that is universal / applicable which can be applied to a wider range of examination timetabling problems (in line with the concern of raising the generality level of the algorithm) that can generate feasible examination schedules (i.e. no conflicting exams are scheduled in the same timeslot)

- To propose optimization method (improvement phase) which will guarantee to improve the quality of the schedules (generated in the construction phase) in terms of maximizing the gap between consecutive exams taken by individual students to allow students to have more revision time between exams, by maintaining feasibility.

Since in this research study, besides aiming to produce feasible schedule (by satisfying had constraint), we are looking at maximizing the gap between consecutive exams taken by students, thus Carter's evaluation function (Carter *et al.*, 1996) was deliberately selected to measure the quality of the examination schedules generated.

1.3 Research Contributions

A summary of the contributions of this thesis are as follows (details are presented in Chapter 6):

- **Reduced complexity of the problem domain.** The *Domain Transformation Approach* proposed has transformed the examination scheduling problem into smaller problem domains that can always be solved in a reasonable amount of time.

- **Reduction of problem space.** *Pre-processing* of constraints has grouped together certain data which provided very useful information through new data representation which reduced the laborious searching during scheduling.

- **Ensuring feasible solutions**. *Allocation of exams to slots* and *split and merge* procedures successfully created feasible exam schedules (without fail) with encouraging figures in terms of number of slots and cost.

- **Efficiency**. *Backtracking* procedure (Carter *et al.,* 1996) which is an improved algorithm that was proposed and managed to further reduce the number of timeslots of the initial feasible schedule.

- **Optimization procedures**. The *Optimization* stage that consists of three steps: minimization of total slot conflicts, permutation of slots and reassignment of exams were proven to be very effective procedures at optimizing the initial feasible exam schedules. A significant reduction in costs for all datasets was recorded.

- **Robust scheduling framework.** *The proposed framework* in this study is very systematic, efficient, robust and is proven to be very *flexible.* This was demonstrated by the success of substituting other procedures in the framework proficiently, i.e. substituting the existing greedy traditional Hill Climbing with the Late Acceptance Hill Climbing and Genetic Algorithm.

- **Consistent performance**. *Through the avoidance of exhaustive exploration* of the search space which normally deploys random

selection between alternative choices during the optimization process, the approach is capable of generating solutions that are *reproducible* and *consistent*. This feature exhibits that the proposed approach managed to raise the generality of the examination scheduling algorithm, which is universal and applicable to a wide range of university examination scheduling problem.

- **Deterministic optimization pattern.** *Deterministic optimization pattern* obtained for all benchmark datasets is an overwhelming achievement since to the best of our knowledge there are no claims made by other researchers resulting in a deterministic pattern for optimization in the university examination scheduling.

1.4 Thesis Overview

This thesis is presented in 6 chapters. The first chapter presents the introduction, scope and objectives of the research.

Chapter 2 describes the overview of the examination scheduling problem, the scheduling approaches or methods developed and the benchmark datasets used over the years in the scheduling research. Some reviews and surveys done by other researchers in the scheduling literature are presented. The motivations that led to our research are also discussed in this chapter.

In Chapter 3 we elaborate in detail on the Domain Transformation Approach proposed in this study. Throughout this chapter, all the main steps involved in generating feasible and improved schedules are

described, including the steps involved in pre-processing, scheduling and optimizations.

Chapter 4 discusses the overall results and the analysis after applying the proposed methods to the Nottingham, Toronto and the International Timetabling Competition (ITC) datasets.

Optimization in our proposed framework involves minimization of total slots conflicts, permutations of exams slots, and reassignments of exams between slots. Chapter 5 zooms in into one of the component of optimization which is the permutations of exams slots which contributed a big percentage of the overall performance achieved through the optimization process discussed in Chapter 4. In this chapter, we discussed and analysed the effectiveness of incorporating a global search procedure (Genetic Algorithm) into the proposed optimization framework in comparison to our previous incorporation of local search procedure.

In Chapter 6, we conclude the thesis by discussing the contributions of the study to the research community and highlight opportunities for possible future works.

CHAPTER 2

Background and Literature Review

This chapter focuses on providing a background to the examinations scheduling research by introducing relevant definitions for the scheduling and discussing the constraints imposed on this problem, as highlighted in the literature. We also summarize and review various surveys done by other researchers in this area. Later we briefly summarize the algorithmic techniques proposed in this area by providing a timeline of representative methods proposed in the last 40 years, in order to outline a general landscape of the categories of methods available. Next, the benchmark datasets, some pre-ordering strategies, and the most widely-used evaluation functions are discussed in brief. In addition to that, we compare the performances of some selected methods that reported encouraging results. Lastly, we also present the insights and motivations obtained by this background study.

2.1 Background of the Scheduling Research

Scheduling research has attracted researchers since the 1960s, especially from the Operational Research community. Since then, there has been a significant number of research activities in this area and the number is still increasing. Over the years, many researchers have made a number of

insightful contributions to the scheduling literature, as surveyed by Qu *et al.* (2009a).

Most of the methods proposed have reported very encouraging results, stating that the schedules generated really have good qualities; however, it has been reported that not a single method or heuristic is able to consistently solve a broad spectrum of scheduling problems because of the incorporation of problem-specific features in the heuristics (Burke *et al.*, 1994a). This observation calls for more extensive research and study into how to generate good quality schedules consistently.

In the following we provide definitions of the scheduling problem adopted by previous researchers, in order to establish the right context for understanding the prior contributions. We also provide some reviews of a list of publications including surveys conducted by some researchers in this area.

2.1.1 Definition of Scheduling According to the Scheduling Literature

Carter and Laporte (1996) defined the basic problem in examination scheduling as:

"The assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes."

Burke *et al.* (2004c) further defined scheduling or timetabling as follows:

"A timetabling problem is a problem with four parameters: T, a finite set of times; R, a finite set of resources; M, a finite set of meetings; and C, a finite

set of constraints. The problem is to assign times and resources to the meetings so as to satisfy the constraints as far as possible."

In the timetabling context, meetings can be referred to as events where normally involved a meet-up between people at a particular location. A general timetabling problem includes scheduling a number of events for example exams or courses into certain number of periods.

According to Qu *et al.* (2009a), examination scheduling (timetabling) problems can be defined as:

"Exam timetabling problems can be defined as assigning a set of exams $E = e_1, e_2, \ldots e_e$ *into a limited number of ordered timeslots (time periods* $T = t_1, t_2, \ldots t_t$ *and rooms of certain capacity in each timeslot* $C = C_1, C_2, \ldots C_t$*, subject to a set of constraints."*

A more general definition of examination scheduling problems is given below:

The examination scheduling problem is the problem of assigning a set of examinations into time slots over a specific period of time such that it satisfies the hard constraints (and some optional constraints if possible) associated with the available resources.

2.1.2 Constraints in the Examination Scheduling Problems

Normally, the main challenge of the examination scheduling problem is to satisfy a wide variety of constraints. In the scheduling literature, constraints can be classified into two categories; hard constraints and soft constraints (Qu *et al.,* 2009a).

- Hard constraints cannot be violated under any circumstances. For instance, conflicting exams (i.e. exams which involve the same students) cannot be scheduled concurrently. Another example of a hard constraint that needs to be satisfied is the room capacity; i.e. there must be enough space in a room to accommodate all students taking a given exam.

  A timetable that satisfies all the hard constraints is called a *feasible* timetable.

- Soft constraints are not critical but their satisfaction is beneficial to students and/or the institution. An example of a soft constraint is the requirement to spread out the exams taken by individual students so that they have sufficient revision time between the exams for which they are enrolled. Typically, one cannot satisfy all of the soft constraints; thus, there is a need for a performance function measuring the degree of satisfaction of these constraints.

  Some of the key (primary) hard constraints and soft constraints suggested by Qu *et al.* (2009a) are listed in Table 2-1 and Table 2-2 respectively.

Table 2-1: Primary Hard Constraints in the Examination Scheduling Problems

Primary Hard Constraints
1. No exams with common resources (e.g. students) can be assigned simultaneously
2. Resources for exams need to be sufficient (i.e. number of exam participants needs to be below the room capacity; enough rooms for all of the exams)

Table 2-2: Primary Soft Constraints in the Examination Scheduling Problems

Primary Soft Constraints
1. Spread conflicting exams as evenly as possible, or not in x consecutive timeslots or days
2. Groups of exams are required to take place at the same time, on the same day or at one location
3. Exams to be consecutive
4. Schedule all exams, or the longest exams, as early as possible
5. Order (precedence) of exams needs to be satisfied
6. Limited number of students and/or exams in any timeslot
7. Time requirements (e.g. exams (not) to be in certain timeslots)
8. Conflicting exams on the same day to be located nearby
9. Exams may be split over similar locations
10. Only exams of the same length can be combined in the same room
11. Resource requirements (e.g. room facility)

Examination scheduling problems can be categorized as either uncapacitated or capacitated. In the uncapacitated examination scheduling problem, room capacities are not considered, while in the capacitated problem the room capacities are treated as a hard constraint.

2.2 Reviews of Various Surveys in the Scheduling Literature

From the 1980s until recently, several surveys have been undertaken in the area of scheduling, with the approaches or methods used in the literature to produce exam schedules being reported. Schmidt and Strohlein (1980), Carter (1986), Carter and Laporte (1996), Burke *et al.* (1997), Schaerf (1999) and Qu *et al.* (2009a) have conducted surveys and overviews of various methods and strategies applied by researchers to solving scheduling problems. Many of the surveyed methods and approaches have successfully solved the examination scheduling problems and some algorithms/heuristics were reported to work well on particular datasets while others performed better when used with different datasets.

A survey conducted in 1980 by Schmidt and Strohlein (1980) summarized the available methods used to generate examination schedules up until 1979. In 1986 Carter wrote a survey paper that includes all the methods developed in the previous 20 years for scheduling examination sessions. This survey (Carter, 1986) is referenced by many researchers in the scheduling community. Based on both of the surveys mentioned above ((Schmidt and Strohlein, 1980) and (Carter, 1986)), it was reported that the majority of researchers formalized the examination scheduling problem as a graph colouring problem. In Carter (1986)'s study, the graph colouring problem was used to produce a conflict-free schedule by applying graph theory.

Ten years later, the author in the previously mentioned survey, together with the co-author (Carter and Laporte, 1996), produced another

survey paper which focused on the state-of-the-art methods in the 1990s. The authors have defined the examination scheduling problem as the assignment of examinations into slots by rewarding the conflict-free condition. The authors also introduced other soft constraints and new benchmark datasets (Toronto) which are now very widely used and tested by researchers in the examination scheduling area. Based on the graph colouring methods, the authors have classified the scheduling methods into four categories: cluster, sequential, meta-heuristics and the constraint-based method. These methods were implemented and experimented on the Toronto datasets. The authors also implemented the Backtracking process which they initially hypothesized could reduce the number of time slots required to schedule the exams. This hypothesis was proven correct in some datasets. The results for the experiments conducted on the Toronto datasets were presented in the paper and since then, the research community has been challenged to propose other approaches with the objective of improving the quality of the schedules based on the same benchmark datasets documented in the literature.

Another survey paper was published by Bardadym (1996) in the same year as Carter and Laporte (1996) produced their survey report, as mentioned in the previous paragraph. In his survey, Bardadym (1996) classified educational scheduling problems into 5 common types: faculty scheduling, classteacher scheduling, classroom assignment, course scheduling and examination scheduling. According to the author, examination scheduling is the most difficult task, and therefore it was claimed that the scheduling system was first proposed with the existence of computers in the universities.

A survey of the state-of-the-art approaches and automated systems in educational scheduling problems was presented a year later by Burke *et al.* (1997). This survey discussed several major approaches in the scheduling research which included Tabu Search, Genetic Algorithm, Simulated Annealing, Memetic Algorithm and Constraint Logic Programming.

Qu *et al.* (2006) in their survey highlighted that the most studied and researched area of scheduling is educational scheduling; mainly the examination scheduling, and due to this their survey concentrated on this type of scheduling. From this literature, the authors have classified and discussed the available methods used in examination scheduling which are motivated by raising the generality of the approaches: graph heuristics, meta-heuristics, constraint-based methods, multi-criteria techniques, hybridizations, and methods that concerned neighbourhood structures, etc.

Qu *et al.* (2009a) in another survey highlights new trends and key research achievements that have been carried out in the last decade. A widespread survey of the development of the search methodologies and automated systems for examination scheduling was done by the authors. According to Qu *et al.* (2009a), meta-heuristics approaches and their hybridization with other search techniques were found to be implemented quite commonly in the examination scheduling problem. In this survey, the author also claimed that different versions of problem datasets with the same name have been circulating in the scientific research community for the last ten years and this has generated some confusion among the researchers. The authors have made the effort to rename the widely-

studied datasets in order to avoid this confusion. Apart from this, the author also summarized the datasets used by some researchers and reported in the literature.

Another recent survey in educational timetabling was conducted by Pillay (2013). However, this survey was not focusing on the examination timetabling problem, instead it can be considered as the first survey that only concentrated on school timetabling. The survey defined school timetabling and discussed a detailed overview on the proposed methods to generate solutions. Besides that, the author also presented the different hard and soft constraints in the school timetabling problem.

A comprehensive study of educational timetabling, a latest survey paper was published recently by Kristiansen and Stidsen (2013). The authors concentrated on the main educational timetabling problems and highlighted some of the main trends and research achievements within educational planning problems. The authors mentioned that they did not intend to perform any experimental comparison on the different methods used, but only to give an overview of the methods. As claimed by Qu *et al.* (2009a), Kristiansen and Stidsen (2013) concluded that many of the used solution approaches are of some kind of hybridization of multiple heuristics.

2.3 Summary of Algorithmic Techniques in the Scheduling Literature

The general approach to solving the scheduling problems usually consists of two phases, i.e. the *construction* and *improvement* phases (Hertz, 1991).

In the first phase, the *construction* phase, a solution is constructed using a sequential construction algorithm. At this point, the solution can be feasible or infeasible. For an infeasible solution, an adjustment is made in the second phase to make it feasible and for a feasible solution an improvement is attempted to enhance its quality.

Scheduling research actually began with straightforward sequential techniques in the 1960s, as discussed in detail by Qu *et al.* (2006). Later, the emergence of many successful techniques was seen; these can be categorized into several broad categories (Carter and Laporte, 1996; Schaerf, 1999; Burke and Petrovic, 2002; Petrovic and Burke, 2004; and Qu *et al.*, 2009a).

In their survey, Qu *et al.* (2006) made mention of the specialization of the scheduling research into sub-areas of educational scheduling, nurse scheduling, transport scheduling, sports scheduling, etc. However, according to the authors the most studied and researched scheduling problem is that of educational scheduling and in particular, exam scheduling. The survey highlighted families of related heuristics deployed in the solution of scheduling problems which include: graph heuristics, meta-heuristics, constraint-based methods, multi-criteria techniques, hybridizations, and methods that focus on the investigation of neighbourhoods in the solution space.

In this section, we will highlight the key algorithmic techniques that have been successfully applied in the examination scheduling problem. Rather than explaining and summarizing the characteristics and algorithms of each technique in detail, which can be found readily in the

literature (for example; Qu *et al.*, 2006; Qu *et al.*, 2009a etc.), we are taking a different approach in presenting and describing the emergence of these methods over the years.

We have provided a timeline that illustrates a historical lineage of key algorithmic techniques for solving examination scheduling problems, as can be seen in Figures 2.1 to 2.4. Please note that these timeline figures were based on selected methods that are widely used and described (most well-cited) in the literature (up to 2014); therefore, recent methods that are not as well established are not depicted in this diagram. Another important note is that the methods were arranged according to the category. In each category, the name of the method was displayed according to the year it was proposed or used, with the intention of illustrating the progression or origination of each method. Some methods were hybridized or integrated with other methods but, in the interest of clarity, the linkages between these methods were not shown in the diagram since the main objective is to provide a general overview of the methods according to their main categories.

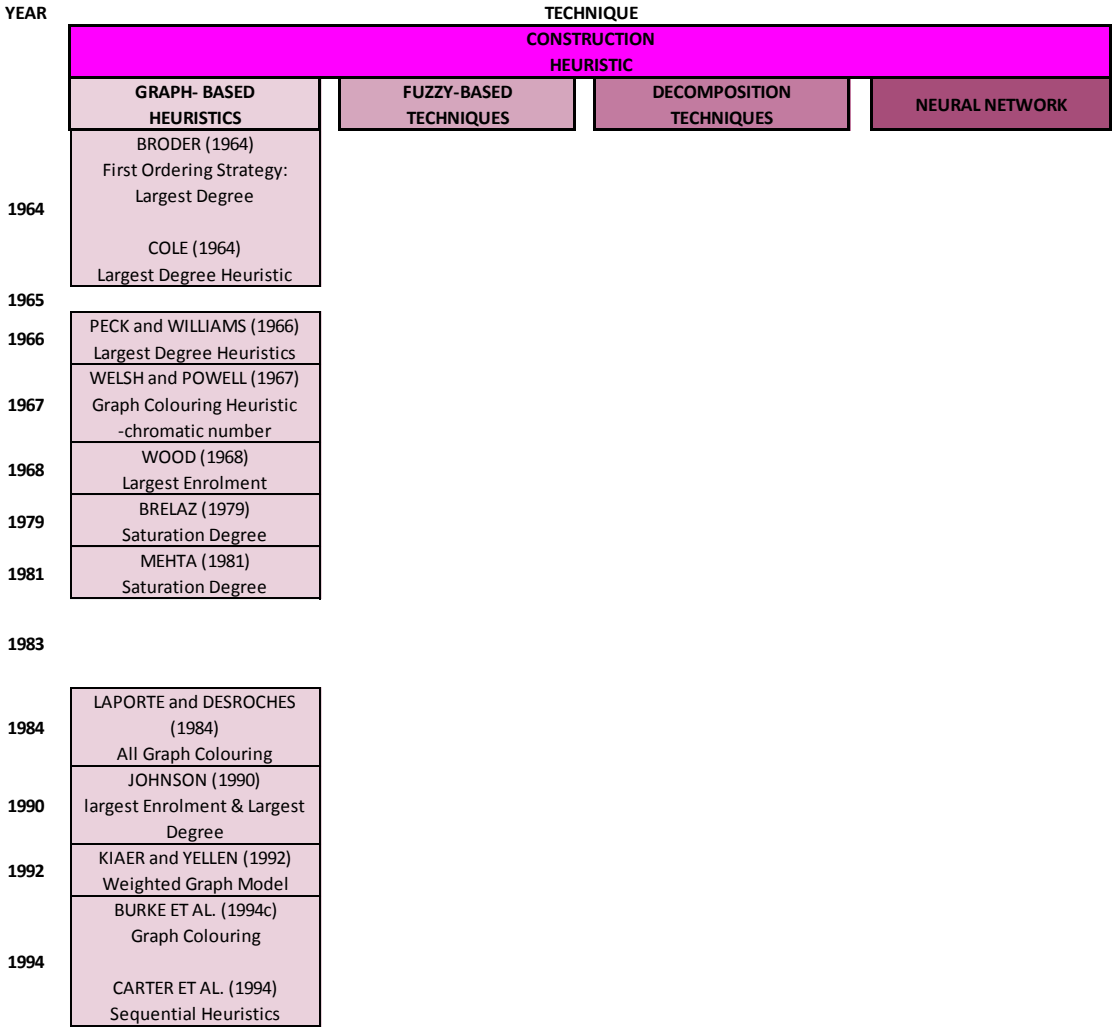


Figure 2.1: Timeline of Brief Historical Lineage of Some Keys Algorithmic Techniques – Constructive Heuristics (1964 – 1994)

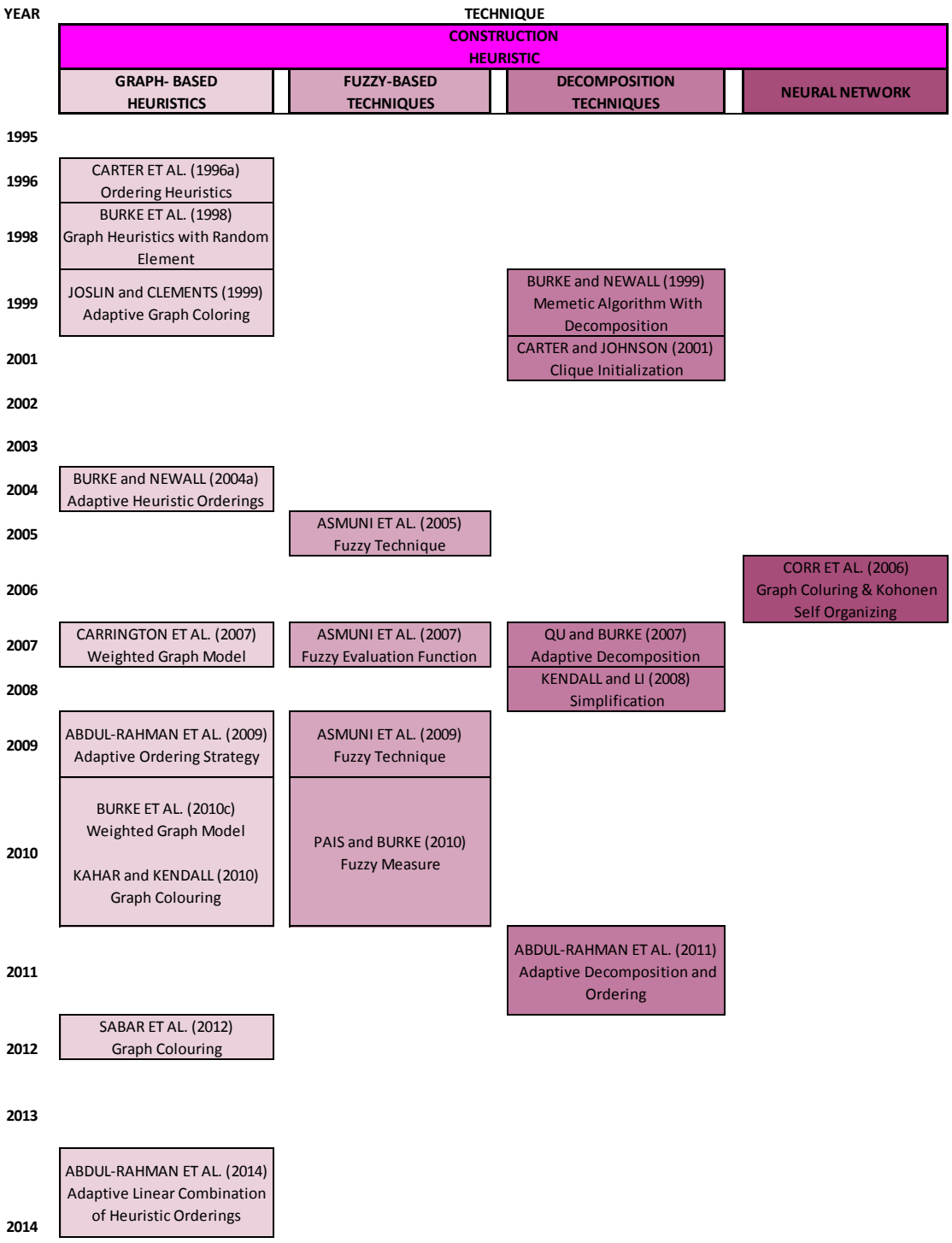


Figure 2.2: Timeline of Brief Historical Lineage of Some Keys Algorithmic Techniques – Constructive Heuristics (1995 – 2014)

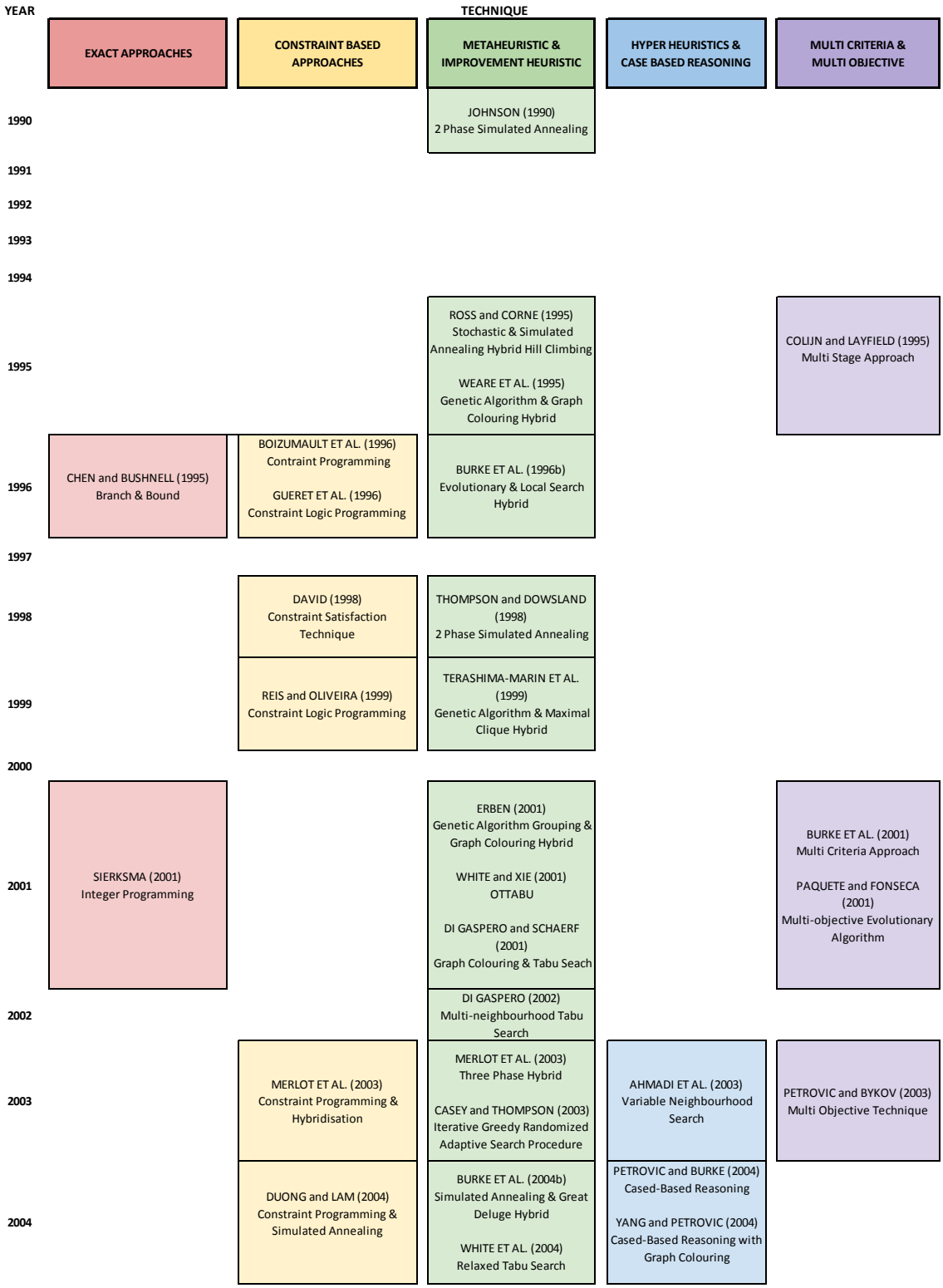


Figure 2.3: Timeline of Brief Historical Lineage of Some Keys Algorithmic Techniques – Various Heuristics (1990 – 2004)

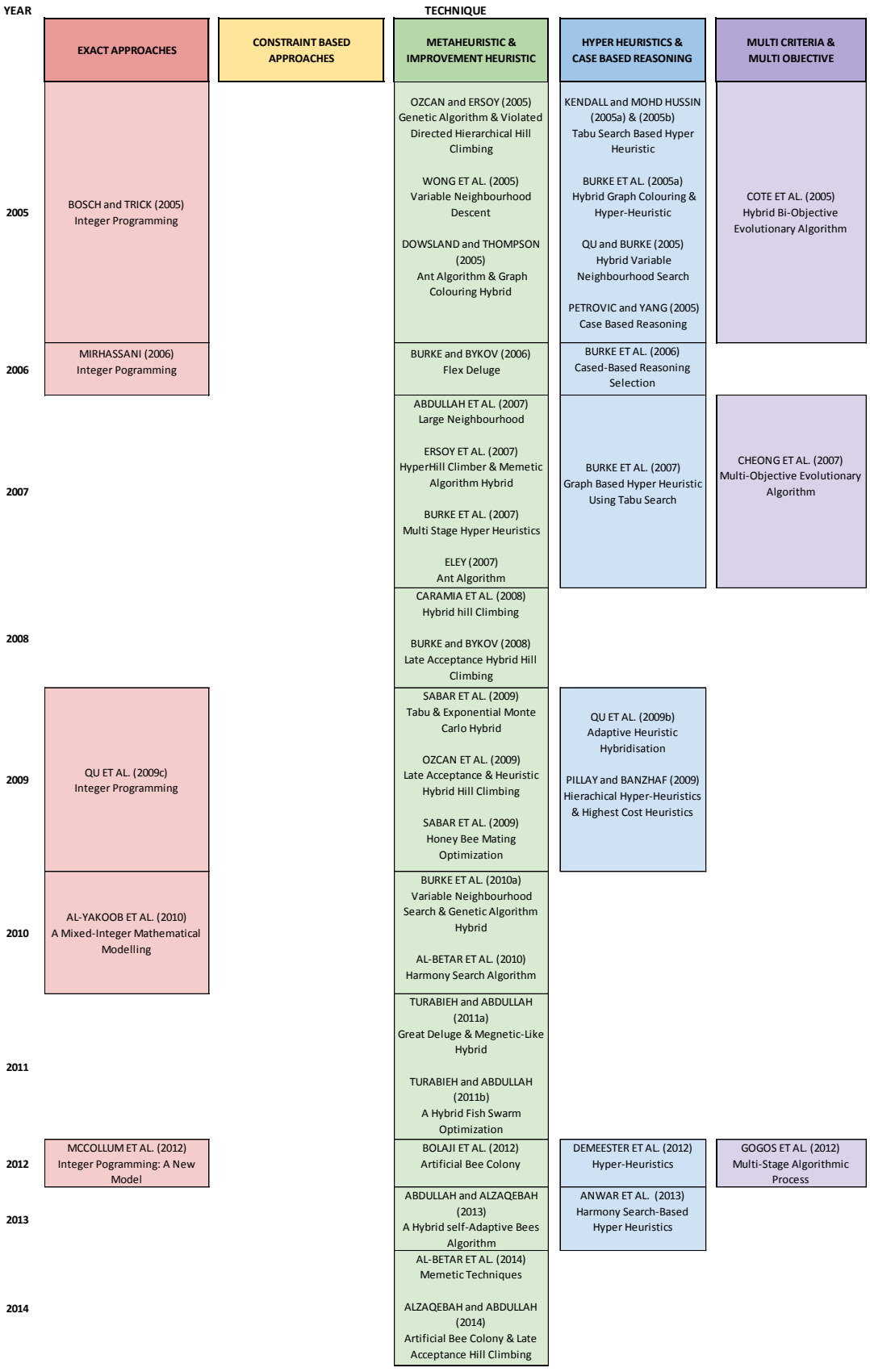


Figure 2.4: Timeline of Brief Historical Lineage of Some Keys Algorithmic Techniques – Various Heuristics (2005 – 2014)

In the timeline diagrams above, several broad categories of techniques used in examination scheduling can be seen. These include constructive heuristics (for example, graph-based heuristics); fuzzy-based techniques; decomposition techniques and neural network. Other techniques include exact approaches; constraint-based; metaheuristic and improvement heuristic; hyper-heuristics and case-based reasoning; and multi-criteria and multi-objective techniques.

Based on the diagrams, we observed that majority of the proposed methods in solving the examination timetabling problems were based on graph-based heuristics and metaheuristic/improvement heuristic techniques, which the latter attracted more interests among the researchers. Despite the rapid emergence or progression of the methods, it was studied that many of the methods are the spin-off or followers of the previous published approaches which did not differ substantially from those established methods.

2.4 Benchmark Examination Scheduling Datasets

From the published research it is clear that benchmark datasets were used quite extensively. The usage of the same standard benchmark datasets in different research conducted by all researchers in this area is very important in order to have a fair judgement about the efficiency and effectiveness of a particular method. Besides, it can also provide a quick understanding and generalization of the strength or capability of a particular method based on the results reported.

In the examination scheduling literature, the most extensively used benchmark dataset is the Toronto dataset proposed by (Carter *et al.*, 1996) which was made publicly available on the internet [ftp://ftp.mie.utoronto.ca/pub/carter/testprob]. The characteristics of all the datasets from Toronto benchmark problems are listed in Table 2-3 in Section 2.4.1. For the Toronto dataset, according to (Qu *et al.*, 2009a) 8 out of 13 problem instances exist in 2 versions. Version I of the datasets which are widely tested by other researchers will be presented in the table.

The data in the table are arranged according to the name of institution, followed by the name of each dataset, number of exams exists in the problem, total number of students registered for the examination session, number of total enrolments of students for the courses, conflict density and lastly required number of exams slots for each dataset.

The Conflict Density represents the ratio between the number of elements of value "1" to the total number of elements in the conflict

matrix. A Conflict Matrix C is a square matrix of dimension number of exams [number of exams x number of exams], and was defined where each element $C_{ij} = 1$ if exam i conflict with exam j (have common students), or $C_{ij} = 0$ if they don't.

Other than Toronto datasets, we include two more datasets, which we will be using in our experimentation phase at a much later stage, ie: the University of Nottingham dataset which could be accessed from [http://www.cs.nott.ac.uk/~rxq/files/Nott.zip] and the International Timetabling Competition 2007 dataset which can be retrieved from [http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php], presented in section 2.4.2 and 2.4.3 respectively. Definitions for column titles for these new tables are the same as given earlier above.

2.4.1 University of Toronto Dataset

Table 2-3: The Characteristics of University of Toronto Benchmark Dataset

Institution	Name of Dataset	No of Exams	No Of Students	No Of Enrolments	Conflict Density	Required No Of Slots
Carleton University	car-s-91 (I)	543	18419	55522	0.14	32
Carleton University	car-f-92 (I)	682	16925	56877	0.13	35
Earl Haig Collegiate	ear-f-83 (I)	190	1125	8109	0.27	24
Ecole des Hautes Etudes Commerciales	hec-s-92 (I)	81	2823	10632	0.42	18
King Fahd University	kfu-s-93	461	5349	25113	0.06	20
London School of Economics	lse-f-91	381	2726	10918	0.06	18
Purdue University	pur-s-93 (I)	2419	30032	120681	0.03	42
Ryerson University	rye-f-92	486	11483	45051	0.08	23
St. Andrews High School	sta-f-83 (I)	139	611	5751	0.14	13
Trent University	tre-s-92	261	4360	14901	0.18	23
University of Toronto, Arts & Science	uta-s-92 (I)	622	21266	58979	0.13	35
University of Toronto, Engineering	ute-s-92	184	2750	11793	0.08	10
York Mills Collegiate	yor-f-83 (I)	181	941	6034	0.29	21

2.4.2 University of Nottingham Dataset

Table 2-4: The Characteristics of University of Nottingham Benchmark Dataset

Institution	Name of Dataset	No. Of Exams	No. Of Students	No. Of Enrolments	Conflict Density
University of Nottingham	Nott (Nottingham a or Nottingham b)	800	7896	33997	0.03

2.4.3 International Timetabling Competition 2007 (ITC2007) Dataset

Table 2-5: The Characteristics of ITC2007 Benchmark Dataset

Name of Dataset	No. of Exams	No. of Students	Required No. of Slots	Conflict Density
Exam1	607	7891	54	0.0505
Exam2	870	12743	40	0.0117
Exam3	934	16439	36	0.0262
Exam4	273	5045	21	0.1500
Exam5	1018	9253	42	0.0087
Exam6	242	7909	16	0.0616
Exam7	1096	14676	80	0.0193
Exam8	598	7718	80	0.0455

2.5 Widely Used Ordering Strategies

In the process of allocating exams to exam slots, researchers have to decide which exam to allocate first to one of the available time slots. With this in mind, various ordering strategies were utilized by researchers (for example; Broder, 1964; Cole, 1964; Peck and Williams, 1966; Welsh and Powell, 1967; Laporte and Desroches, 1984; Burke *et al.*, 1994c; Carter *et al.*, 1994; Joslin and Clements, 1999; Burke and Newall, 2004a; Abdul-Rahman *et al.*, 2009; and Kahar and Kendall, 2010). It was proven that the ordering strategies affect the final outcome and quality of the solution generated (as discussed by Asmuni *et al.*, 2005). In the normal practise in the timetabling literature, most researchers will try out all ordering strategies (to preorder the datasets) and select the strategy that produce the best results. The summary of the widely-used ordering strategies in Graph Heuristics made by Qu *et al.* (2006) is presented in the following table:

Table 2-6: Widely-Used Graph Heuristics in Exam Scheduling

Heuristics	Ordering Strategy
Saturation Degree	Increasingly by the number of timeslots available for the exam in the timetable at the time
Largest Degree	Decreasingly by the number of conflicts the exams has with other exams
Largest Weighted Degree	This is the same as Largest Degree but weighted by the number of students involved
Largest Enrolment	Decreasingly by the number of enrolments for the exam
Random Ordering	Randomly ordered exams
Color Degree	Decreasingly by the number of conflicts the exam has with those scheduled at the time

2.6 Widely-Used Evaluation Function: Carter Evaluation Function

The standard objective of examination scheduling that is widely used in the literature is to minimize the cumulative inconvenience implied by the temporal proximity of consecutive exams taken by individual students. Based on this objective, in order to have a good quality timetable, each exam to be taken by a student should be scheduled as far apart as possible from one another. The quality of the timetable is measured by the cost function originally proposed by Carter *et al.* (1996) as in the Equation (2.1) below:

$$\frac{1}{T}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} s_{ij}\, w_{|pj - pi|} \tag{2.1}$$

where N is the number of exams, s_{ij} is the number of students enrolled in both exams, i and j, p_j is the time slot when exam j is scheduled, p_i is the time slot when exam i is scheduled and T is the total number of students. Based on this cost function, a student taking two exams that are $| p_j - p_i |$ slots apart, where $| p_j - p_i | = \{1, 2, 3, 4, 5\}$, leads to a cost of 16, 8, 4, 2, and 1, respectively. The lower the cost obtained, the higher the quality of the schedule, since the gap between two consecutive exams allows students to have extra revision time.

It is worth noting here that the gap of the consecutive exams taken by individual students that are more than 5 slots apart (i.e. 6 and above), will not have any penalty, therefore the cost will be zero. According to Carter cost function (Equation 2.1), if all consecutive exams taken by all students in the problem are scheduled 5 slots apart, then the timetable is

considered a zero cost timetable (but this is very seldom since in real life it will cause a very long duration of examination session).

2.7 Performance of Methods Proposed in the Examination Scheduling Literature

In order to analyse the effectiveness of the available methods proposed in producing feasible examination schedules, we have presented the results in terms of the Carter cost (2.1) produced by some researchers and compiled by Abdul-Rahman *et al.* (2011) and Qu *et al.* (2009a). The results are presented in three different tables according to the categories of the methods; i.e. constructive, hyper-heuristics, and numerous improvement approaches on the Toronto datasets. Note that the first column of these tables contains the name of each dataset in the Toronto benchmark problem as can be found in Table 2-3 of this thesis.

Table 2-7: Comparison of Results in Terms of Carter cost (2.1) for the Thirteen Problem Instances of Toronto Benchmark Datasets For Different Constructive Approaches Reported in the Literature

Problem	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
car-s-91 (I)	7.10	**4.97**	5.45	5.29	5.08	5.03	5.18	5.08
car-f-92 (I)	6.20	4.32	4.50	4.54	4.38	**4.22**	4.44	4.34
ear-f-83 (I)	36.40	36.16	36.15	37.02	38.44	**36.06**	39.55	38.28
hec-s-92 (I)	**10.80**	11.61	11.38	11.78	11.61	11.71	12.20	11.13
kfu-s-93	**14.00**	15.02	14.74	15.80	14.67	16.02	15.46	14.42
lse-f-91	**10.50**	10.96	10.85	12.09	11.69	11.15	11.83	11.43
pur-s-93 (I)	**3.90**	-	-	-	-	-	4.93	5.74
rye-f-92	**7.30**	-	-	10.38	9.49	9.42	10.04	9.37
sta-f-83 (I)	161.50	161.90	**157.21**	160.40	157.72	158.86	160.50	157.34
tre-s-92	9.60	8.38	8.79	8.67	8.78	**8.37**	8.71	8.73

uta-s-92 (I)	3.50	**3.36**	3.55	3.57	3.55	3.37	3.49	3.52
ute-s-92	**25.80**	27.41	26.68	28.07	26.63	27.99	29.44	26.24
yor-f-83 (I)	41.70	40.77	42.20	39.8	40.45	**39.53**	42.19	40.38

[1]-(Carter and Laporte,1996), [2]-(Burke and Newall, 2004a), [3]-(Qu and Burke, 2007), [4]-(Asmuni *et al.*, 2009), [5]-(Abdul-Rahman *et al.*, 2009), [6]-(Burke *et al.*, 2010c), [7]-(Pais and Burke, 2010), [8]-(Abdul-Rahman *et al.*, 2011)

Table 2-8: Comparison of Results in Terms of Carter cost (2.1) for the Thirteen Problem Instances of Toronto Benchmark Datasets For Different Hyper-Heuristics Approaches Reported in the Literature

Problem	[9]	[10]	[11]	[12]	[13]	[14]
car-s-91 (I)	5.37	5.36	**4.97**	5.16	5.17	5.19
car-f-92 (I)	4.67	4.53	4.28	**4.16**	4.32	4.31
ear-f-83 (I)	40.18	37.92	36.86	35.86	**35.70**	35.79
hec-s-92 (I)	11.86	12.25	11.85	11.94	11.93	**11.19**
kfu-s-93	15.84	15.20	14.62	14.79	15.30	**14.51**
lse-f-91	-	11.33	11.14	11.15	11.45	**10.92**
pur-s-93 (I)	-	-	**4.73**	-	-	-
rye-f-92	-	-	9.65	-	-	-
sta-f-83 (I)	157.38	158.19	158.33	159.00	159.05	**157.18**
tre-s-92	**8.39**	8.92	8.48	8.60	8.68	8.49
uta-s-92 (I)	-	3.88	3.40	3.42	**3.30**	3.44
ute-s-92	27.60	28.01	28.88	28.30	28.00	**26.70**
yor-f-83 (I)	-	41.37	40.74	40.24	40.79	**39.47**

[9]-(Kendall and Hussin, 2005a), [10]-(Burke *et al.*, 2007), [11]-(Pillay and Banzhaf, 2009), [12]-(Qu and Burke, 2009), [13]-(Qu *et al.*, 2009b), [14]-(Burke *et al.*, 2010e)

Table 2-9: Comparison of Results in Terms of Carter cost (2.1) for the Thirteen Problem Instances of Toronto Benchmark Datasets For Other Different Improvement Approaches Reported in the Literature

Problem	[15]	[16]	[17]	[18]	[19]	[20]	[21]
car-s-91 (I)	5.10	**4.50**	5.40	5.20	6.60	4.60	4.80
car-f-92 (I)	4.30	3.93	4.20	4.40	6.00	**3.90**	4.10
ear-f-83 (I)	35.10	33.71	34.20	34.90	**29.30**	32.80	34.92
hec-s-92 (I)	10.60	10.83	10.40	10.30	**9.20**	10.00	10.73

kfu-s-93	13.50	13.82	14.30	13.50	13.80	**13.00**	**13.00**
lse-f-91	10.50	10.35	11.30	10.20	**9.60**	10.00	10.01
pur-s-93 (I)	-	-	-	-	**3.70**	-	4.73
rye-f-92	8.40	8.53	8.80	8.70	**6.80**	-	9.65
sta-f-83 (I)	157.30	158.35	157.00	159.20	158.20	**156.90**	158.26
tre-s-92	8.40	7.92	8.60	8.40	9.40	7.90	**7.88**
uta-s-92 (I)	3.50	**3.14**	3.20	3.60	3.50	3.20	3.20
ute-s-92	25.10	25.39	25.30	26.00	**24.40**	24.80	26.11
yor-f-83 (I)	37.40	36.53	36.40	36.20	36.20	**34.90**	36.22

[15]-(Merlot *et al*., 2003), [16]-(Yang and Petrovic, 2004), [17]-(Cote *et al*., 2005), [18]-(Abdullah *et al*., 2007) [19]-(Caramia *et al*., 2008), [20]-(Burke *et al*., 2010a), [21]-(Turabieh and Abdullah, 2011a).

The results presented in the above three tables are arranged according to the 13 Toronto datasets problem proposed by Carter *et al.* (1996). These results were obtained by some of the researchers using numerous techniques. Each column consists of the Carter cost (2.1) for each dataset in this Toronto benchmark problem.

According to the Carter cost (2.1), we could say that the cost is actually the average penalty of the students spread in the examination schedule. An achievement of a zero cost timetable means that the timetable is of a very high quality, and we can imagine that every single student will have at least a five slots' gap between one exam and the next in the examination session.

However, none of the costs obtained and reported in the examination scheduling research on the Toronto benchmark problem have a zero cost (as can be seen in the above three tables), which means that in real life some of the inconvenience is tolerated in order to achieve a shorter examination period.

In the three tables presented above, each bold value is the best value for each dataset reported among the researchers. Overall, the costs obtained are considered to be very encouraging, as the lowest Carter cost (2.1) obtained is 3.14 for dataset uta-s-92 (I). Here the value 3.14 is the value of the average penalty of the students spread in the examination schedule.

It is worth noting here, however, that the listed methods have a rather uneven performance. They perform well against some benchmark problems and less well against others. One important point to note when comparing the performance of the various methods is that several of the best results have been obtained by methods that did not report any results for some datasets; for example, for lse-f-91, pur-f-93 (I) and rye-f-92.

2.8 Pre-Processing Approach in the Examination Timetabling

Based on the observations of Table 2-7 to 2-9, there are quite a number of approaches that are unable to produce results for certain benchmark datasets, which after analysis we can determine that the inability to produce feasible solutions for a problem is due to the size and complexity of the relationships among the entities in the problem space. For example, by analyzing datasets lse-f-91, pur-f-93 (I) and rye-f-92, we observed that these problems have a high ratio value of number of exams against required number of slots (as can be seen in the last column of Table 2-10). The ratios are 21.17, 57.60 and 21.13 for lse-f-91, pur-f-93 (I) and rye-f-92 respectively which means that on average these are the

minimum number of exams to be allocated per slot. The higher this value is, the harder it is to find the exams that are not conflicting among one another.

Table 2-10: No of Exams to Required No of Slots Ratio

Name of Dataset	No of Exams	No Of Students	No Of Enrolments	Conflict Density	Required No Of Slots	No of Exams to Required No of Slots Ratio
car-s-91 (I)	543	18419	55522	0.14	32	16.97
car-f-92 (I)	682	16925	56877	0.13	35	19.49
ear-f-83 (I)	190	1125	8109	0.27	24	7.92
hec-s-92 (I)	81	2823	10632	0.42	18	4.50
kfu-s-93	461	5349	25113	0.06	20	23.05
lse-f-91	381	2726	10918	0.06	18	21.17
pur-s-93 (I)	2419	30032	120681	0.03	42	57.60
rye-f-92	486	11483	45051	0.08	23	21.13
sta-f-83 (I)	139	611	5751	0.14	13	10.69
tre-s-92	261	4360	14901	0.18	23	11.35
uta-s-92 (I)	622	21266	58979	0.13	35	17.77
ute-s-92	184	2750	11793	0.08	10	18.40
yor-f-83 (I)	181	941	6034	0.29	21	8.62

We foresee that there is a need to minimize or reduce the complexity of the problem or we hypothesize that what if we were to transform the problem into another problem where there is a possibility that the complexity of the existing problem can be degraded into simpler problems. To enable this, an understanding of the data is required, in line

with this notion we observe an approach by Thomas *et al.* (2009) which tries to give a better understanding of the problem space to the timetable designer has a merit in which by understanding the correlation of all the entities in the problem space a solution can be generated.

Thomas *et al.* (2009) approached the timetabling problem by introducing a pre-processing stage that visualized the timetabling data. The researches were confident that the visualization will provide a new insight or analysis of the timetabling data that would help the timetable designer and decision maker to formulate a feasible timetable. The researchers used Prefuse which is a Java-based extensible software framework for pre-processing to visualize the data. They provided five interaction techniques to the users to interact with the data, namely Selection, Explore, Encode, Filter and Connects. Selection, enables the marking of a particular data that can be further analysed. Explore, enables the visualization of the timetabling data to be interacted, showing a different perspective or concentrating only on a specific part of the problem space. Encode, enables the user to change the visual representation of the data. Filter, enables the user to add certain restrictions on the data to be visualized enabling the user to focus on certain part of the data. Connects allows the user to view interconnected data within the problem space. The pre-processing stage provides additional interactions to the scheduler (person) on the interrelation or linkage of all the elements in the problem domain. The pre-processing stage through visualization enables the timetable designer to learn more about the data and with this knowledge it is hoped it would help the timetable designer to design a better timetable.

There is also another approach by Gunawan *et al.* (2008) which provides another insight where the pre-processing of data to generate new representation of information can be utilized within the algorithm to help in constructing a better quality timetable. Gunawan *et al.* (2008) proposed a hybrid approach which combines Tabu Search and Simulated Annealing to solve the teacher and course scheduling simultaneously. The approach consists of three phases; the pre-processing stage, initial construction stage and the improvement stage. The initial construction stage concentrates on finding the initial feasible timetable.

The researchers constructed new information which is the information on which teacher is willing to teach a particular course, resulting in a set of new data connecting a particular paper with the probable teacher. The information was generated from the preferences given by the teachers. The second information generated is the list of slots that a particular teacher prefers to teach which is given by the day and time period. These two lists are generated and sorted based on the preferences set by the teachers.

Gunawan *et al.* (2008) reported that the pre-processing is done on the information of preferences provided by the teachers, which is actually considered as the soft constrains of the actual problem. The main problem (scheduling) is being solved using the greedy heuristics (similar to Gunawan *et al.* (2007a)) without the assistance on the new information generated. This opens up a new avenue where the pre-processing can be conducted on the data related to the hard constraints. New information can be generated which will give a new representation that will enable the algorithm to understand the problem space.

What interests us, we observe that these two papers (Thomas *et al.* (2009); Gunawan *et al.* (2008)) which touched on pre-processing, did that specifically and without the intention to want to alter the data representation of the problem space. Hence, the intention that we have is to provide an alternative methodology that transforms the problem space into a different representation that could open-up new avenues or simplify the problem to a more manageable and deterministic solution. This is with the understanding that many of the researchers claim that the exam timetabling is an NP-complete problem which requires huge amount of resources to fully explore the entire search space of a feasible solution and more over to find the best solution within these feasible timetables.

2.9 Important Insights from the Scheduling Literature and Motivations for the Research

Despite many methods having been proposed to date to solve the examination scheduling problems, various findings have concluded that there is no single heuristic that is able to solve all scheduling problems effectively (Burke *et al.*, 1994). Meta-heuristics approaches - for example, Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS) etc., which were believed to generate promising results - were improved further through the introduction of hyper-heuristic approaches (Qu *et al.*, 2009a).

Notwithstanding the advantages and capabilities of the many methods reported in the literature, we are aware that the results for some problems are not easily reproducible because most of the algorithms

depend on some random number generation. These algorithms deploy random selection between alternative choices during the optimization process. This means that a simple change in the generation of random numbers may affect very significantly the direction of the optimization process. As a result, the randomness generates different results. This makes the results only statistically comparable. Since the results are hard to reproduce, it is difficult to determine whether they are optimal or not.

A huge volume of publications have reported the investigation and refinement of hyper-heuristics. Various methods concerning the design and selection of heuristics and hyper-heuristics have been proposed and evaluated. On one hand, there have been various improvements in the examination schedules produced using these methods. On the other hand, this suggests that the results generated in this way cannot be seen as definitive.

We have also learned from the background study that some researchers have classified the examination scheduling problem as an NP complete problem (e.g. Cooper and Kingston, 1996; and Even *et al.*, 1976). An NP complete problem is a problem which cannot be resolved to a global optimum in a reasonable amount of time. Currently, with the flexibility of the students' enrolments, there was a great increase in size of the examination timetabling problem, which also has increased the complexity of this problem (McCollum, 2007). As the examination scheduling problem is classified as an NP complete problem, it can be understood that the resources needed to solve the problem grow very rapidly with the size of the problem. Hence, some problems cannot be solved even on the fastest computers, and in the examination scheduling

context, it means that the optimal schedules are not generated successfully and one has to accept sub-optimal (but feasible) solutions.

It is worth emphasizing that the examination scheduling problem represents a challenging computational problem due to the strong interactions between the many-to-many relationships between the data of students and exams. The challenge and complexities of the problem increase when most of the universities allow flexibility for the students to register on modular courses across faculties (Burke *et al.*, 1994). The increasing size of students' enrolments and different choices of available courses increases the challenge and complexity of this real-world problem (McCollum, 2007).

From the background study we can learn that some methods that deploy random selection between alternative choices during the optimization process failed to reproduce the solutions obtained previously. This is because a simple change in the generation of random numbers may affect very significantly the direction of the optimization process, thus generating different solutions. This means that the results produced with methods deploying random selection are only statistically comparable and cannot guarantee the quality of every individual solution.

All of the above scenarios and phenomena create motivations for further research. In general, the literature review and background study have provided insights into the following:

a) a new approach to analyzing the complex system by looking at different levels of abstraction;

b) abstraction of essential features in order to simplify the data used in scheduling by doing pre-processing of data and constraints;

c) propose a definite step (a constructive approach) to schedule the exams to ensure the method can reproduce the schedule at any time;

d) sub-dividing the problems into smaller sub-problems in order to reduce the NP complexity of the examination scheduling problems as described in the literature, and therefore increase the efficiency in terms of the computational time;

e) the exploration of the search space that is guided by one heuristic which avoids exhaustive exploration of the search space.

CHAPTER 3

Domain Transformation Approach to Examination Scheduling

This chapter presents the proposed framework for solving examination scheduling problems. We start by giving an overview of the Domain Transformation Approach – the approach that transforms the original problem domain into different and smaller domains which are easier to manage. We provide the general framework proposed in this study, which consists of several main stages; namely, the pre-processing of data, scheduling and optimization. Each step is then elaborated in greater detail by providing the algorithm, its essential elements and its computational complexity.

3.1 Domain Transformation Approach – Overview

Classical description of examination scheduling implies a search in a large solution space which is typically accomplished with the aid of heuristics to control the exploration of the search space. We propose that the transformation of the problem domain is an effective methodological approach to dealing with complex examination scheduling problems. In the proposed approach, we define alternative data structures that capture the

essential dependences in the examination scheduling problem. By performing an appropriate pre-processing of the original student-exam data into suitable data structures, we can map the original problem expressed in the multi-dimensional space of exams and students into a space with a reduced dimensionality of exams and exam-slots. We will refer to this approach to solving the scheduling problem as the *Domain Transformation Approach.*

Domain Transformation Approach therefore could be defined as an approach whereby a problem is transformed into a simpler problem expressed in terms of different variables from the original problem description. Examples of the domain transformation approach in other application areas include the subdivision of a problem domain into multiple sub-problems (e.g. the Danzig-Wolfe decomposition for solving linear programming problems), transformation of problem variables (e.g. the Fourier Transform, employed to transform signals between time or spatial domain into frequency domain) and the transformation from continuous to discrete functional description (e.g. the Z-transform converting time domain signals into discrete domain of trains of pulses), to mention just a few prominent examples.

The proposed domain transformation of the examination scheduling focuses on the pre-processing of constraints prior to the generation of a feasible timetable. This is done through the abstraction of essential features of the exam scheduling problem from the original student-exam data. This data abstraction process constitutes a significant methodological contribution of this study, as it enables subsequent optimization of the examination schedule without the need to refer to the voluminous student-

exam data in the course of the optimization. One example of a pre-processing is the identification of the clashing exams. This information will ease and expedite the scheduling process later because less permutations are needed to obtain this information since it is readily available. Unlike other approaches, without employing pre-processing, a lot of permutations are needed, since this information is implicit in data. Other examples of pre-processing will be discussed in further detail in this chapter later.

This approach was inspired by insights from previous studies on industrial process optimization (Bargiela, 1985; Argile *et al.*, 1996; Peytchev *et al.*, 1996; and Bargiela *et al.*, 2002) and has been formalized as a Granular Computing methodology (Pedrycz *et al.*, 2000; Bargiela and Pedrycz, 2002; Bargiela *et al.*, 2004; and Bargiela and Pedrycz, 2008).

Granular Computing is an emerging conceptual and computing paradigm of information processing methodology (Pedrycz *et al.*, 2000), (Bargiela *et al.*, 2002), (Bargiela *et al.*, 2004), (Bargiela and Pedrycz, 2008). In the concept of Granular Computing, the key element is multiple levels of information processing sometimes called hierarchical processing. Each level will perform different types of processing that will result in different types of information representation or meaning. In general, Granular Computing can be viewed as human inspired paradigms of computing and information processing (Pedrycz *et al.*, 2000; Bargiela and Pedrycz, 2002; Bargiela *et al.*, 2004; Bargiela and Pedrycz, 2008).

According to Granular Computing concept, the information processing will create information granules and this process is known as Information Granulation (Bargiela and Pedrycz, 2002). According to

Merriam-Webster's Dictionary (http://www.merriam-webster.com), a *granule* is defined as *"a small particle; especially: one of numerous particles forming a larger unit"*. These information granules, with regard to Granular Computing concept, are collection of entities that are arranged together due to some criteria, and normally they are central to the abstraction processes in solving many tasks.

Information Granulation (Bargiela and Pedrycz, 2002) serves as an important medium to simplify problem that needs to be split into smaller sub tasks. It provides an abstraction mechanism that reduces the overall conceptual burden in the original problem space. By having different sizes or representations of the information granules, certain amount of details can be hidden during the problem solving. This offers advantage in terms of reducing the complexities of the problems. As we can imagine, the consistent existence of some details are sometimes unwelcome because they complicate things and therefore they need to be hidden.

As far as the examination scheduling problem is concerned, Granular Computing problem solving strategy could be applied successfully to produce feasible and good quality exams schedules. The systematic approach which involves information processing will create new data representation which will provide valuable and meaningful information that could definitely ease the scheduling task.

Granular Computing in scheduling involves analyzing or representing the scheduling problem at various levels of abstraction. For example, at the fine resolution we may deal with individual students taking individual exams (which is a standard problem definition) as

illustrated in Figure 3.1, at the coarser resolution we deal with classes of exams (for example non-conflicting exams) and formalise the problem description using these classes as illustrated in Figure 3.2. The implication of this is that we deal with several complementary problem descriptions at different levels of generality or accuracy. The more general descriptions serve to facilitate an approximate problem solution in a smaller search space and more detailed representations preserve the possibility of refinement of the solutions. This approach contrasts with the standard, detailed level of problem representation which requires deployment of various heuristic methods to cope with computational complexity.

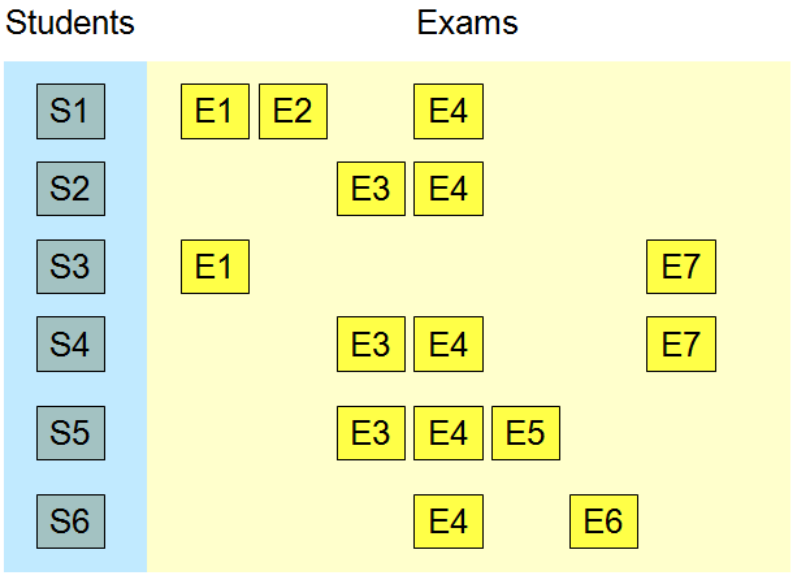


Figure 3.1: Illustration of an Example of a Standard Examination Scheduling Problem (Fine Resolution Level)

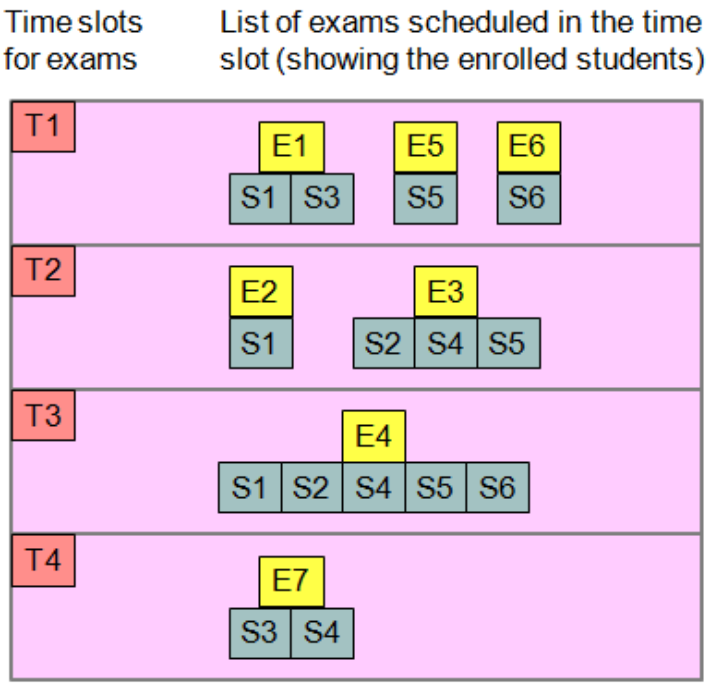


Figure 3.2: Illustration of Classes of Exams - Group of Non-Conflicting Exams With the Students Enrolled (Coarser Resolution)

The key hypothesis of this thesis is that the pre-processing of initial problem data can lead to a transformation of the scheduling problem into a new solution space in which the problem is solved more easily. This aggregated data from the modified data space which are grouped appropriately will be much easier to handle, as opposed to dealing with the original data, as has been done in many previous studies.

We also argue that after applying pre-processing, scheduling could be done more efficiently, generating reproducible results.

3.2 The Flow of the Proposed Approach

This research is proposing a different approach from the work done by others who utilized pre-processing methods; for example, Gunawan *et al*. (2007b), who used a hybrid algorithm which consists of three phases: (1) pre-processing, (2) construction, and (3) improvement in the teacher assignment-course scheduling problem. The pre-processing phase in their work involves assigning teachers to courses by sorting them in descending order, based on their preferences towards the course.

In the approach advocated in this thesis, the aim is for the pre-processing method on the timetable datasets to be employed before the real scheduling process is undertaken. Possible data will be combined in the datasets in such a way that will satisfy the hard constraints imposed on the timetable. These combinations include the courses, rooms and students. Each pre-processing stage will lead to a richer representation and collection of data containing more information to make the final scheduling easier. The revelation of dependencies existing within the data at the aggregated level, which may be difficult to handle at the detailed level, is the fundamental rationale behind the information granulation and subsequent Granular Computing (Bargiela and Pedrycz, 2002). It is postulated that the pre-processing will improve the efficiency and ease of the scheduling task because only feasible solutions will be available to work with, since the pre-processing eliminates all unfeasible timetables from the solution space. The flow of the proposed work is given below:

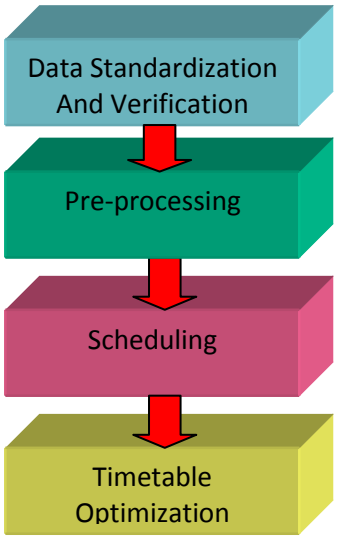


Figure 3.3: The Flow of the Proposed Approach

The steps of the proposed work in creating feasible and quality examination schedules are: standardization and verification of the problem description data, pre-processing, scheduling and lastly, timetable optimization, as illustrated in Figure 3.3.

The above figure clearly shows that in order to produce feasible and good quality examination schedules, the very first step is to do a standardization and verification of the original data files (timetabling problem). Once this is done, pre-processing of data files will follow to generate meaningful aggregated data construct that will ease the next task which is the scheduling. In the scheduling stage, exams will be assigned to slots, which always ensure the feasibility of the schedules. Despite the feasibility of the schedules, the initial orderings of exams produced by the scheduling stage might not be optimal (because it might not fulfil certain soft constraints), therefore this requires a separate deployment of optimization process to further improve the quality, hence the need of the last stage, the optimization. In this final stage, the schedules cost will be minimized via certain procedures.

3.2.1 Standardization and Verification of the Problem Description Data

The first step in this proposed approach is to perform the standardization and verification of the problem description data. The standardization and verification of data are done on the examination scheduling benchmark datasets retrieved earlier that are freely made available to the public over the internet. These data will be used to produce the information shown in Figure 3.9, Figure 3.10: and Figure 3.11.

In the early stage, the datasets that were used are the benchmark exam scheduling data for the University of Nottingham, semester 1, 1994 – 95 and University of Toronto, as presented in the previous chapter. The files contain information pertaining to students, exams, enrolments and data (other data and constraints). This information will be retrieved and assigned to a data representation format that would be easy for future processing. At the same time, there is the concern of Lewis (2008) regarding the disadvantage of heavy reliance on certain benchmark datasets. Consequently, the proposed approach has also been tested on other benchmark datasets from the International Timetabling Competition 2007 (ITC2007).

The datasets produced and made available by the researchers come in various representations and formats. The variations come from the representations of information about courses, students and classes made available in the datasets. For example, for University of Nottingham dataset, there is a student-exam enrolment data representing a list where each row contains a ten characters alphanumeric student ID (or code) and

eight characters exam code as depicted in Figure 3.4. Each student will have a number of rows depending on the number of exams the student has enrolled. For instance, the first five rows of data in the figure represents that the student with student ID *'A890186790'* was enrolled for five exams with exam code: *'R13001E1', 'R13006E1', R13016E1', 'R13021E1'* and *'R13022E1'*.

Unlike the Nottingham dataset, for the Toronto dataset, the enrolment file consists of rows containing a variable-length list of four digits exam code. Each row represents exams enrolled by a particular student. This can be seen in Figure 3.5. If we observe this figure, we can see that the student code is not supplied in the file. Based on the list given in this figure, we can view that the first student in the list (assume that student id = *'1'*) is enrolled for one exam only which is exam with the code *'0174'*. The other two students, in the second and third row were enrolled for exam '*0329'* and *'0332'* respectively. The list continues with the fourth student enrolled for exam *'0377', '0378', '0392'* and *0406',* and the list continues for other students in the dataset. It **i**s worth highlighting here that these two data files are totally in different format, thus need to be standardized and verified in the initial stage.

Some researchers represent the courses in the form of course codes and some in the form of unique numbers – this is also the case with the information about students and classes. Initially, a solution was developed for one dataset with the intention to later provide a more generic algorithm that would cater for various kinds of datasets formats and arrangements.

```
A890186790 R13001E1
A890186790 R13006E1
A890186790 R13016E1
A890186790 R13021E1
A890186790 R13022E1
A891097581 R23102E1
A891097581 R23107E1
A891097581 R23111E1
A891097581 R23117E1
A891097581 R23209E1
A892884191 Q4B102E1
A892884227 AA3008E1
A892884294 V73232E1
A892884294 V73MD1E1
A892884308 Q3A308E1
A892884308 V62200E1
A892884319 V73128E1
A892884331 V73232E1
A892884331 V73MD1E1
A892956502 V73232E1
A892956502 V73MD1E1
A892956513 V73232E1
A892956513 V73MD1E1
A892992407 V73232E1
A892992407 V73MD1E1
A892992496 V73232E1
A892992496 V73MD1E1
A903428732 R13001E1
A903428732 R13003E1
A903428732 R13012E1
A903428732 R13021E1
A903549679 V73127E1
A903549679 V73128E1
```

Figure 3.4: Sample of Enrolment Data from the University of Nottingham Dataset File

```
0174
0329
0332
0377 0378 0392 0406
0528 0531 0532
0014
0164
0008
0226 0229
0355
0110 0236
0291
0049 0156 0292 0303 0319
0304
0049
0403
0061 0103 0158
0249
0005
0297 0299
0239 0241 0242 0262
0353
0205 0422
0061 0062
0071
0353
0240 0241
```

Figure 3.5: Sample of Enrolment Data from the Toronto Dataset File

Recall that we have also decided to test our approach on the ITC2007 dataset. In this particular dataset, in contrast to the Toronto dataset which is in the perspective of students, the ITC2007 is however is in the perspective of exams. A sample of the ITC2007 data file is illustrated in Figure 3.6. Each row represents an exam, where it consists of a two or three digit numbers showing the duration of the exam in minutes. The information in each row is then followed with a variable-length list of a one digit up until four digits student code for all students enrolled for this exam.

```
180, 312, 752, 760, 768, 858, 879, 1920, 1987
180, 312, 752, 760, 768, 858, 879, 1920, 1987
180, 390, 838, 893, 894, 1344, 1491, 1786, 1923, 2066
180, 390, 838, 893, 894, 1344, 1491, 1786, 1923, 2066
180, 378, 384, 841, 860, 862, 883, 1914, 1978, 2099, 2258
180, 378, 384, 841, 860, 862, 883, 1914, 1978, 2099, 2258
180, 728, 840, 847, 877, 882, 1272, 1573, 1784, 1791, 1919, 2062
180, 728, 840, 847, 877, 882, 1272, 1573, 1784, 1791, 1919, 2062
180, 447, 709, 845, 1790, 1980, 1982
180, 379, 382, 844, 867, 1325, 1575, 1788, 1916, 2199
180, 379, 382, 844, 867, 1325, 1575, 1788, 1916, 2199
180, 388, 391, 836, 851, 853, 874, 875, 876, 1475, 1507, 1915
180, 388, 391, 836, 851, 853, 874, 875, 876, 1475, 1507, 1915
180, 365, 849, 854, 865, 869, 1408, 1574, 2321, 5431, 5497
180, 365, 849, 854, 865, 869, 1408, 1574, 2321, 5431, 5497
180, 380, 750, 766, 837, 846, 857, 861, 1143, 1330, 1792, 1903,
180, 380, 750, 766, 837, 846, 857, 861, 1143, 1330, 1792, 1903,
180, 7, 40, 710, 868, 870, 881, 1466, 1552, 2384
180, 7, 40, 710, 868, 870, 881, 1466, 1552, 2384
```

Figure 3.6: Sample of Enrolment Data from the ITC2007 Dataset File

In the above diagram, by assuming that both the first and second row in the list represent exam with 180 minutes duration, if we observe these two rows, we could see that there are 8 students (same students) with student ID: *'312', '752', '760', '768', 858', '879', '1920'* and *'1987'* were enrolled for these two exams.

The main algorithm, as presented below, was designed to utilize a specific data type to represent the scheduling data. It was decided to use matrix as the main data type to represent all the information pertaining to the scheduling problem in the solution space. Since the matrix data type is highly adaptable in terms of the complexity of the representation in the sense that it can easily be converted from a single dimension to two dimensions and so on, this robustness only requires minimal changes in the actual program coding to be implemented. In this study a few matrix

or data types were identified that will be used to keep the initial data and also processed data within the system.

The main data type is the *StudentExamList matrix* that represents the relationship between a student and all the exams that the student will be required to sit. It is a matrix of dimension *NumberOfStudents x MaxNoOfExamForAStudent + 1*. This data structure will be used to generate other data representations of the problem space. Each row index will represents a student, the first column will contain the total number of exams that the students have registered. Subsequent column will contain the examination index. The *StudentExamList* will be supported by the *ExamLookupIndex* and *StudentLookupIndex*. The *ExamLookupIndex* is a matrix of *NumberOfExam x 2*. Each row in the *ExamLookupIndex* will hold information for an exam. The first column contains the actual exam code or name and the following column will contain the number of unique students sitting for the exam. Similar to *ExamLookupTable*, the *StudentLookupTable* holds information for a student. Each row represents a student. The first column stores the student's actual ID Number and the second column holds the number of exams the students will be sitting in. The relations of these data structures can be seen in the following algorithm.

The algorithm to alleviate the initial problem of dataset and format variety is by providing an algorithm or function that would convert a dataset format to a standard format that will be used as an input to the pre-processing stage. The algorithm consists of three subroutines each for a particular dataset, namely Nottingham, Toronto and ITC2007 dataset.

The Nottingham subroutine will extract information from the input file. The Nottingham input file consist of rows with two column of information, The Student ID and the Exam ID each of this piece of information will be converted to an integer value reference. The unique. reference id for an exam and student will be used to populate the *StudentExamList*. While placing the exam id in the *StudentExamList* this subroutine will also keep the count of exams a student is enrolled and the number of students sitting for a particular exam. Once the placement of all the information is completed, a verification function will be called to verify all the information in the *StudentExamList* is exactly the same is the information in the original file. The verification will also check if there are inconsistencies in the input file.

The Toronto subroutine is responsible to read and convert information from the input file to the format that is required by the scheduling algorithm. Each row in the Toronto input file is the list of exams a student is enrolled in which is deliminated by spaces. The Toronto file does not provide any information on the student id thus requiring the subroutine to assume that the first list of exams belongs to student with id equals to 1 and so on until the end of the file. The algorithm will place the exam id on the *StudentExamList*, keeps the tally for the number of exams a student is taking and the number of students sitting for a particular exam.

The ITC2007 subroutine on the other hand will have to read and filter information in the input files as part of the data is not being used in our implementation. Each row in the ITC2007 dataset file has the duration of an exam and the list of student id enrolled in the exam

deliminated by a comma. Since the dataset does not provide any exam id, the subroutine will assume that the first entry in the dataset belongs to exam id equals to 1 and so on. Similar to the previous two routines, this routine will also populate the *StudentExamList*, keeps track of the number of exams a student is enrolled in and tally the number of students sitting for a particular exam.

Algorithm 1

```
If Nottingham Dataset
        Open the Data File
                While not End Of File
                        Read a line from file to Input
                        Get FirstToken from Input  //StudentID
                        Get SecondToken from Input  //ExamID
                        i = -1
                        j = -1
                        Find Index of SecondToken in ExamLookupIndex assign to i if
                        found
                        Find Index of FirstToken in StudentLookupIndex assign to j if
                        found
                        If j = = -1
                          LastSLI = LastSLI + 1
                          StudentLookupIndex[LastSLI] = FirstToken
                          j = LastSLI
                        EndIf
                        If  i = = -1
                            LastELI = LastELI + 1
                             ExamLookupIndex[LastELI] = SecondToken
                             i = LastELI
                            StudentExamList[j][( StudentExamList [j][0])+1] = i
                        StudentExamList [j][0]= StudentExamList [j][0]+ 1
                        Else
                             StudentExamList [j][( StudentExamList [j][0])+1] = i
                            StudentExamList [j][0]= StudentExamList [j][0]]+ 1
                        EndIf
                        UpdateLookupTable(StudentLookupIndex,j,
                            ExamLookupIndex,i)

                End While
        Close
        DataValidity = VerifyData(ExamLookupIndex, StudentLookupIndex)
End if

If Toronto Dataset
        Open the Data File
                While not EndOfFile
```

```
                    i = i+1;
                     j=0;
                    Read a line from file to Input
                    While Input not empty
                        j=j+1
                        Get FirstToken from Input //Space Deliminated
                        StudentExamList [i][j]=FirstToken
                        UpdateLookupTable(StudentLookupIndex, i,
                      ExamLookupIndex, FirstToken)

                    End While
                    StudentExamList[i][0]= j
                End While
        Close
        DataValidity = VerifyData(ExamLookupIndex, StudentLookupIndex)
End if

If ITC2007 Dataset
        Open the Data File
                i = 1;
                While not EndOfFile
                        Read a line from file to Input
                        Get FirstToken from Input  //Exam Duration,not used
                        j = 0
                        While Input not empty
                            Get FirstToken from Input  //Comma Deliminated
                            j= j+1
                          StudentExamList[FirstToken][
                                StudentExamExam[FirstToken][0] ] = i
                          StudentExamExam[FirstToken][0]= StudenExamList
                                [FirstToken][0]+1
                         UpdateLookupTable(StudentLookupIndex,FirstToken,
                        ExamLookupIndex,i)

                        End While
                i = i + 1
                End While
        Close
        DataValidity = VerifyData(ExamLookupIndex, StudentLookupIndex)
End if
```

Figure 3.7: Algorithm for Retrieving Enrolment Data, Standardization and Verification

3.2.2 Pre-processing

A key step in the proposed exam scheduling method is the pre-processing of constraints prior to the generation of a feasible timetable. This is done through the abstraction of essential features of the exam scheduling problem from the original student-exam data.

One example of the information obtained from the pre-processing is the identification of the clashing exams. Due to the need to ensure the feasibility of timetables, typical timetabling algorithms check if exams do not clash every time an exam is scheduled. In other words, for conventional approaches, without the pre-processing stage, the clashing information is implicit in data; thus, a lot of permutations requiring a lot of time need to be done in order to create a feasible timetable. This problem can be avoided using the approach of this study. The data structure is part of the mechanism to ensure that the feasibility of all generated schedules is maintained. By devising a data structure combining non-clashing exams into separate entities one can avoid subsequent feasibility checks. The data structure enables easy lookup of exams that can be scheduled together. We take an example of exam A, if exam B is in the non-clashing list of exam A, then they can be scheduled together. Otherwise there is at least one student that is enrolled in exam A and exam B. Hence, this approach deals only with feasible solutions.

The pre-processed data can also be utilized later to find another information in the pre-processing stage; for instance, the non-clashing exams information, all exams will have its corresponding non -clashing list. To find the non-clashing exams, we just need to focus solely on the

clashing exams information logically, by finding the inverse of the clashing exams. This means that instead of doing a lot of cross-checking and cross-referencing across many files, we are only employing the information that we obtained through the previous pre-processing. At each stage of the next level of pre-processing we will be doing a hierarchical processing that will always provide us with richer information. The types of pre-processing mentioned above are just examples. Other types of pre-processing and data dependencies will be considered to further enrich the existing information in order to minimize and simplify the scheduling process, thereby creating a valid and optimal exam timetable.

The pre-processing stage has generated the following information:

1. Number of students for each exam.
2. List of students in each exam.
3. List of clashing exams for each exam.
4. List of non-clashing exams for each exam.
5. Generation of the exam-conflict matrix.
6. Generation of the conflict chain.
7. Generation of the spread matrix.

Generation of the Exam Conflict Matrix

The first pre-processing step is to determine potential clashes between examinations and to count the number of students causing these clashes. This information is used to construct an exam conflict matrix which is a square matrix of dimension equal to the number of exams. Entries in this matrix at position *(i,j)* represent the number of students causing conflict

between exams *i* and *j*. The exam conflict matrix is generated by incrementing the value at position *(i,j)* by 1 for each student taking exams *i* and *j* when the student-exam list is traversed. The matrix will contain a negative number of students value (-*s*) at position *i,j* if there are *s* students causing conflict between exams *i* and *j*. The exam conflict matrix is a static data representation of the problem space. Information contain herein is fixed, which represent the interrelation between an exam to another exam. It forms the reference for allocation, optimization and calculation of the schedules quality (Carter cost (2.1)). The algorithm to generate this matrix is given in Figure 3.16.

Generation of the Conflict Chains

The clashes between exams are static information or relation which will not change in a problem space. By this we mean that the exam clashes will only change with an addition of a student taking both exams or all the students taking the two exams drop or unregister for either one of the exam. A clash between two exams is a situation where there is one or more students taking the two exams, thus implies that the two exams cannot be scheduled concurrently. This representation provides useful information granules that can be utilized in the scheduling process. Based on these information granules we determine the minimum number of time slots that are necessary for scheduling the given set of examinations. We refer to this stage as the construction of conflict chains.

The algorithm deployed at this stage can be summarized as follows:

1. Initiate the algorithm by allocating all exams to time slot one.

2. Select the first exam as “current” and initiate the counter for the current conflict chain.

3. Label the current exam as “allocated to the current chain” and note all of the exams that are in potential conflict with the current exam.

4. If the list of potentially conflicting exams is non-empty, re-allocate those exams to the next available time slot. Otherwise, label the current chain as complete and proceed to Step 6.

5. If the list of potentially conflicting exams is non-empty, select the first exam from the list and repeat from Step 3 with the currently selected exam.

6. Check if all exams allocated at Step 1 are belonging to one of the conflict chains; if YES, then the algorithm terminates; if NO, then the conflict chain counter is incremented and the unallocated exam is taken as “current” for processing, starting from Step 3.

Figure 3.8: Algorithm to Generate Conflict Chains

In this section we will illustrate the generation of conflict chains based on an example data. Assuming that Figure 3.9 is the student-exam list that was generated after the standardization and data retrieval phase. We are using four (4) students that have enrolled in total of 7 exams. The exam-students list generated will be as in Figure 3.10.

This information will be used to generate the Exam Conflict Matrix, resulting in a conflict matrix in Figure 3.12. Note that the contents of the Exam Conflict Matrix are negative values. Each value is derived from the number of students that enrols in an exam from the x-axis and the y-axis.

Example we have two students taking exam E1 and E24 which is student A and B.

The conflict chains generation as illustrated in Figure 3.13 starts by assigning all the exams to the first slot (i.e. slot number 1). Next the algorithm will traverse the exam list that has been assign to slot 1. It will start with the first exam and marking it as assign to slot 1. It will then check all other exams in slot 1 against the accepted exam to determine if it clashes (utilizing the exam clash list in the process). E1 has been marked as accepted and the algorithm will check E1 with the rest of the content of Slot 1. E24 is in the clash list of E1 thus marked as clash and it will be shifted to the next slot (slot 2) in the shifting phase, same goes to E300, E45 and E60.

Upon completion of exam E1 inspection, the algorithm will mark the second exam which is still unmarked or not allocated; the slot still contains E512 and E73. E512 is marked as accepted and the algorithm will inspect E512 against E73 which will result in marking E73 as clash and to be moved to the next slot. Upon completion of E512 inspection the algorithm will mark another exam in Slot 1 as accepted, however Slot 1 currently does not contain any exams unallocated, hence marking the completion of the checking phase.

In the next phase all exams that were marked as “to be shifted” will be shifted to the second slot, the exams are E24, E300, E45, E73 and E60. The checking cycle continues by accepting E24 and evaluating its clash with other exams in Slot 2. E300 and E45 will be marked as to be shifted. E73 will then be marked as accepted and E73 clash list will be inspected

and no exam is being marked as to be shifted. Finally E60 will then be marked as accepted and E60 clash list will be inspected and no exam is being marked as to be shifted. In the subsequent shifting phase, E300 and E45 are being shifted and the process continues until all the exams are accepted.

Once the process of generating conflict chains has been completed, the algorithm will check the maximum number of slots obtained against the maximum slot required for a dataset. If the value of current slot configuration is lower than the maximum slot required, the exam in the last slot will be separated to create another slot as illustrated in Figure 3.13 (After N Process). The final exam to slot allocation is depicted in Figure 3.14.

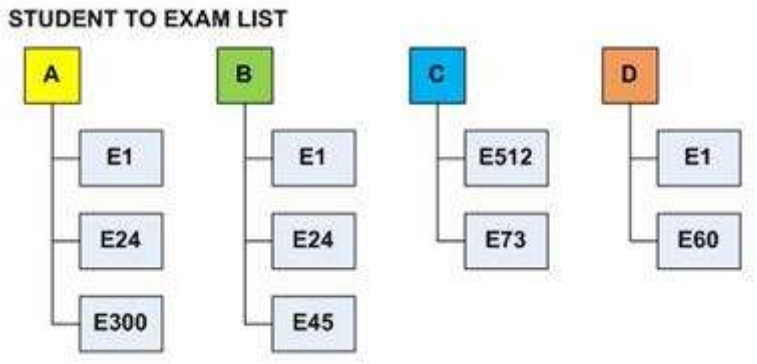


Figure 3.9: An Example of a representation of Student-Exam List

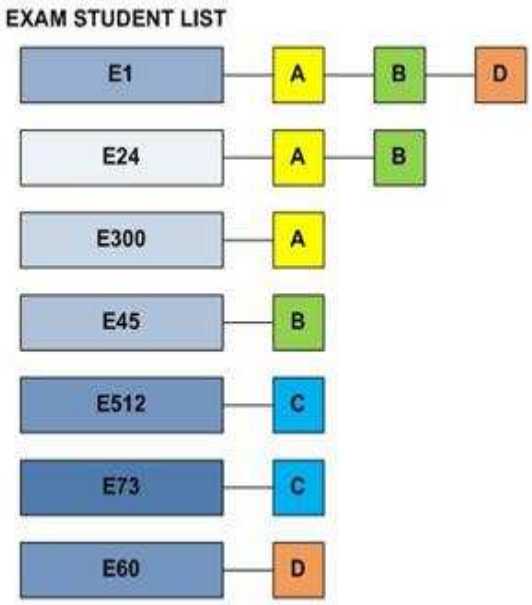


Figure 3.10: Exam-Students List Generated Based on the Student-Exam List

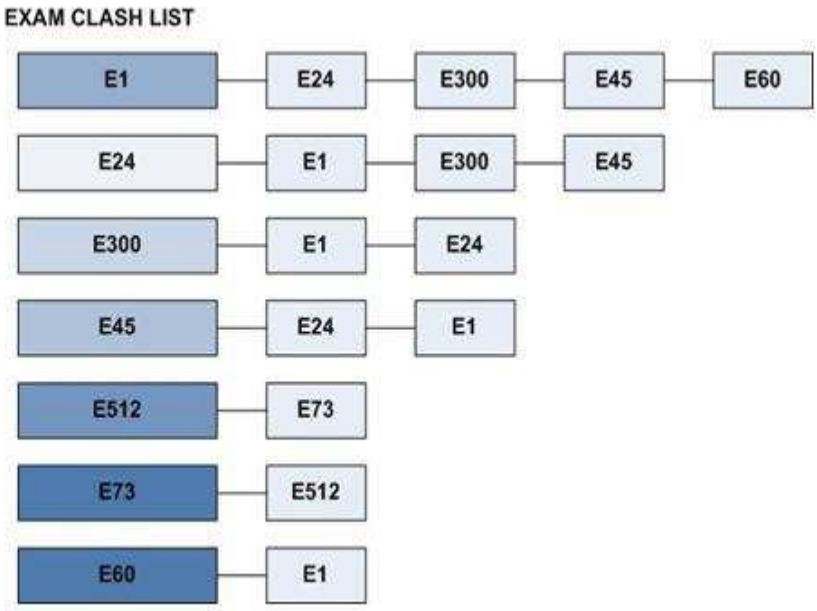


Figure 3.11: Exam-Clashes List

EXAM CONFLICT MATRIX

	E1	E24	E45	E60	E73	E300	E512
E1	0	-2	-1	-1	0	-1	0
E24	-2	0	-1	0	0	-1	0
E45	-1	-1	0	0	0	0	0
E60	-1	0	0	0	0	0	0
E73	0	0	0	0	0	0	-1
E300	-1	-1	0	0	0	0	0
E512	0	0	0	0	-1	0	0

Figure 3.12: Illustration of Exam-Conflict Matrix

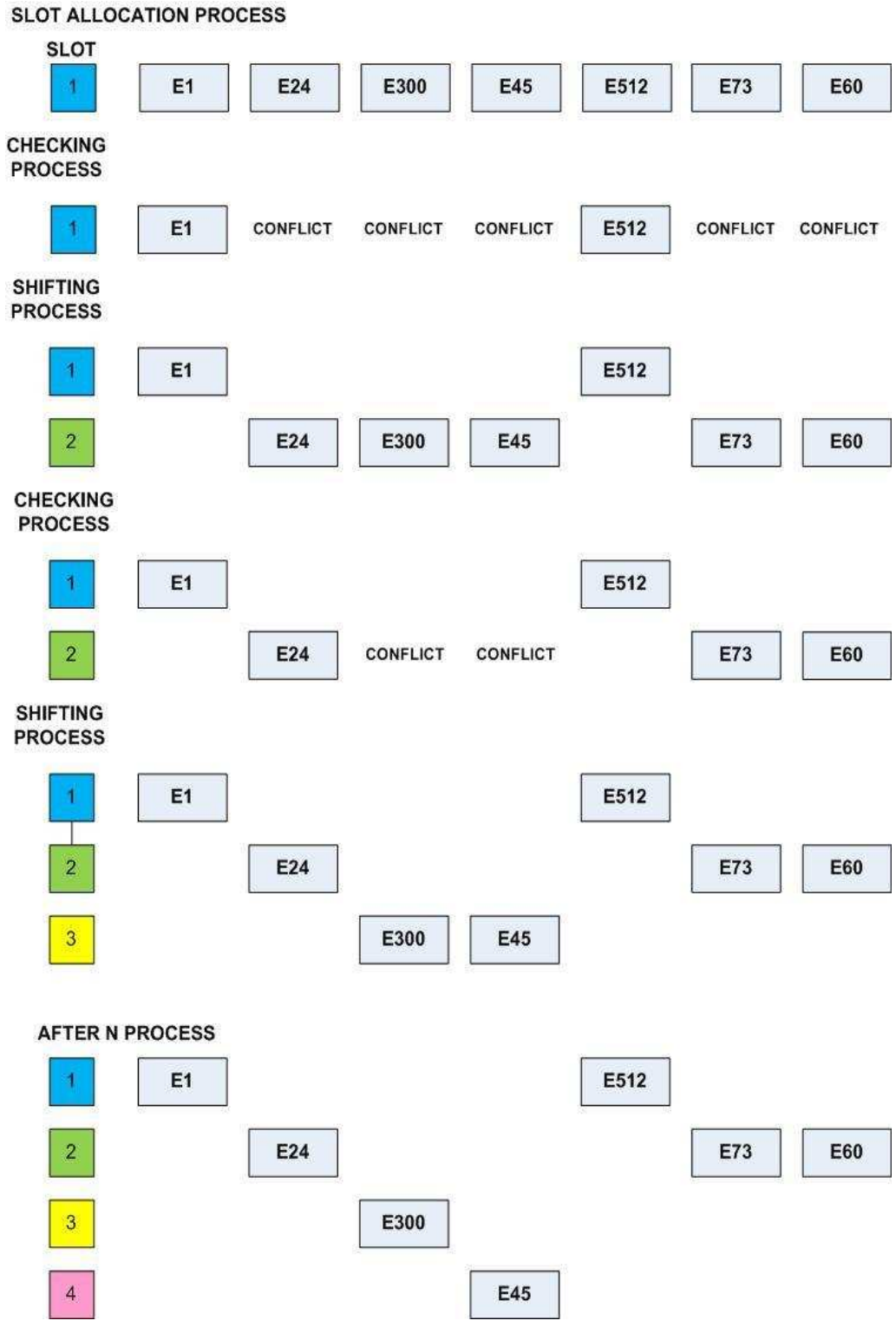


Figure 3.13: Diagram Illustrating the Slot Allocation Process

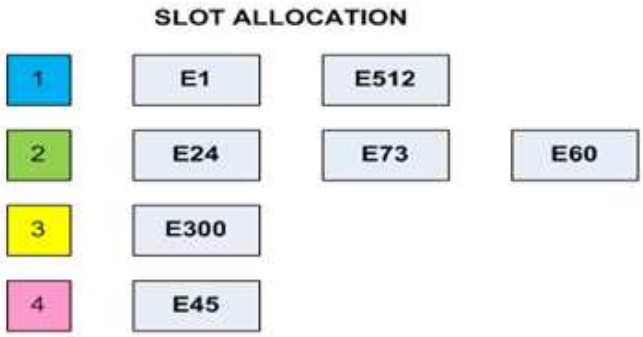


Figure 3.14: Diagram Illustrating Exams Allocated To Slots

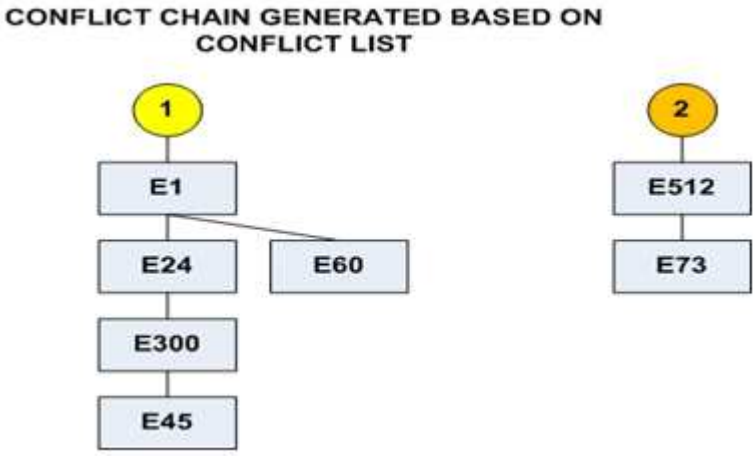


Figure 3.15: Conflict Chains Generated

The outcome of the above algorithm is a set of conflict chains that represent mutually dependent exams that need to be scheduled in different time slots in order to avoid the violation of hard constraints (Figure 3.14). However, the algorithm implies that it is possible to have one exam belonging to more than one conflict chain (although the algorithm will ensure that the allocation of this exam to the time slot is consistent in both chains). For this reason we perform the additional step of merging these conflict chains, which happen to have common exams.

The merged conflict chains represent independent subsets of the examination set that can be dealt with one at a time.

Generation of the Spread Matrix

Besides generating the independent conflict chains, as outlined above, the number of students who take exams allocated to time slots that are 1, 2, 3, 4 and 5 time slots apart was evaluated. Since we are dealing with information granules that represent a potential conflict between all exams in one time slot and all exams in another time slot, regardless of what the actual time slot numbers are, we create a framework for efficient optimization of the cost function (measuring the quality of the timetable). The following will describe the proposed scheme for renumbering the time slots using the background knowledge about the structure of the cost function. This stage will be referred to as maximizing the spread of examinations.

Using the exam conflict matrix information together with initial grouping of exams information through the early pre-processing stage, the spread matrix is then generated. The spread matrix (Rahim *et al.*, 2009) is a square matrix of dimension S, where S is a number of slots. Entries in the spread matrix at position (p,q) represent the number of students who take an exam from both slot p and slot q. The matrix is symmetrical with diagonal elements being omitted because students can take only one exam in any given exam slot. The spread matrix is created by incrementing the value at position (p,q) by 1 if exam p and exam q are not grouped together in the early allocation process (meaning they are clashing).

The pre-processing of the original student-exam data into the exam conflict matrix and the spread matrix pays dividends in terms of minimizing the subsequent cross-checking and cross-referencing in the original data in the optimization process, thus speeding up the scheduling task. The essence of pre-processing is summarized by the pseudocode in Figure 3.16.

Algorithm 2

```
Read student-exam list
Initialise exam-conflict matrix to zero
Initialise spread matrix to zero
Initial allocation of exams to slots
Read exam-to-slot allocation vector
For i=1 to number-of-students
  For j=1 to number-of-exams-of-student-i -1
   For k=j to number-of-exams-of-student-i
    Increment entry exam-conflict(student-exam(j),student-exam(k)) by 1
     If exam-to-slot(student-exam(j))/=exam-to-slot(student-exam(k))
     THEN
     Increment matrix element spread(j,k) by 1
    End
   End
  End
End
```

Figure 3.16: Algorithms for Pre-Processing

The pre-processing stage is one of the biggest contributions towards solving and minimizing the search space. In the approach that is proposed and implemented in this study, the granulation of the problem space was introduced using the exam-conflict matrix, spread matrix and exam-to-slot vector to simplify the problem and provide an algorithm which is not NP complete to solve the problem. The main computational component in the algorithm is the outer loop which iterates through the student list, l which ranges between 611 to 30032 based on the three benchmark datasets used in this study as can be found in Chapter 2. For each of the students there is an inner loop to create a permutation of the exams that the students are

taking, m with itself to create the exam-conflict matrix and spread matrix. The value of m has a limitation, which is actually the maximum number of exams a student can enrol in a particular semester. By assuming that one exam is equivalent to a one credit hour, a worst case scenario, a student will enrol for a maximum of 25 exams. The number of exams m is selected from a pool of exams ranging from 81 to 2419 based on the benchmark datasets used in this study. The complexity of the algorithm can be simplified to $O(l \times m \times m) = O(lm^2)$. Within the problem domain when l is increased its relative value towards m is huge making m irrelevant. The value of m can be neglected due to the fact that m has a limit to its value, which is very small compared to the number of students l when it grows. Thus, the complexity of the algorithm is simplified to $O(l)$.

The pre-processing of data and constraints from the original problem space will provide important information granules which in turn provide valuable information for scheduling. The new aggregated data generated by the pre-processing stage, i.e. ***exam conflict*** *and* ***spread matrices,*** *will minimize the subsequent cross-checking and cross-referencing in the original data in the optimization process, thus expediting the scheduling process.*

3.2.3 Scheduling

After the pre-processing of data is completed, the next step is the scheduling process. This is when the initial allocation of exams to slots is done, i.e. grouping exams that are not conflicting in a group. In this study,

there are two methods for scheduling; the first is via the conflict chains generation and the second is via the allocation method.

3.2.3.1 Scheduling for Uncapacitated Problems

Scheduling will be done using the derived information from the pre-processing stage. The timetable generated at this stage is based on pre-processed data; therefore, it will always fulfil the hard constraints.

The generation of a feasible solution is achieved using an allocation method which is based on the standard Graph Colouring Heuristic (Broder, 1964), (Cole, 1964), (Peck and Williams, 1966), (Welsh and Powell, 1967), (Laporte and Desroches,1984), (Burke *et al.*, 1994c), (Carter *et al.*, 1994), (Joslin and Clements, 1999), (Burke and Newall, 2004a), (Asmuni *et al.*, 2007), (Abdul-Rahman *et al.*, 2009), (Kahar and Kendall, 2010), which is used to generate the allocation of exams to time slots. This method allocates exams by placing exams with the highest conflicts first; it then moves to other exams with lower conflicts. This is based on the principle of an early allocation of those exams with the highest number of conflicts to the available time slots. During this process, the number of conflicts of exams which have not been scheduled yet is recalculated to reflect the latest updated status of exams. This means that all unallocated exams are taken into consideration in every iterative step, rather than being processed sequentially.

During the allocation of exams to slots, there will always be two types of slots: empty slots and non-empty slots. Empty slots are the slots are not yet been assigned any exams, where as non-empty slots are the

slots that already have exams been assigned to them. We have four preferences for allocation determination which are: assigning conflicting exams to non-empty slots; assigning conflicting exams to empty slots; assigning non-conflicting exams to non-empty slots; and assigning non-conflicting exams to empty slots. These have the values of 0.4, 0.3, 0.2 and 0.1, respectively. The higher the value, the higher the preference for allocation.

Any unused slots are removed and provide a buffer-space for subsequent optimization. The output is an allocation flag, exam-to-slot vector which contains the slot numbers for all exams. An allocation flag is a single dimensional array or also known as a column vector of dimension [number of exams x 1] where each value in the vector corresponds to the slot number where each exam in problem is assigned. At this point, the number of slots could be determined by the maximum value in the allocation flag.

The generation of a feasible solution (or what can be considered here as a feasible conflict chain) is done by allocating a group of exams to timetable slots which are verified by calling a verification procedure. The process continues by calling the merging procedure to reallocate exams. By splitting a slot *p* and reassigning constituent exams to other slots, the total number of slots may be reduced if every exam in slot p can be allocated to some other slot, i.e. is not in conflict with exams in other slots.

Algorithm 3

```
Generate a feasible allocation of group of exams to timetable slots
Verify allocation of exams to slots
Execute splitandmerge procedure
        Split a slot p and reassign constituent exams to reduce the number of slots
```

Execute backtracking to further reduce number of slots

Figure 3.17: Algorithm for Allocation of Exams to Time slots

The generation of a feasible solution process through the allocation of exam to timetable slots in Algorithm 3 is further detailed in Algorithm 3a.

Algorithm 3a

```
Create the first slot, islot=1;

Initialize allocflag array, xs array and inew to 0.
initialize xe with the exam conflict matrix
while there is still exam unallocated
        if inew > 0 there was a new assignment to allocflag
                update 'xe'
        Obtain an unscheduled exam id (istart) with the biggest conflict
        if the obtained exam has a confnum==0
                assign all exams not yet allocated with value nex + 1
        if exam 'istart' can't be allocated to 'islot-1' slots
                allocate istart to the last slot 'islot'
                update xs with the latest exam
                increment islot by 1
                update xs with the new slot availability
        else
                assign exam istart with value nex + 1 indicating deferred assignment
        inew=istart
end
initialize inew and xc matrix to 0
reinitialize xe with the exam conflict matrix
for i=1 to number of exams
        if exam i is allocated to nex+1
                Assign ye the number of conflicts of exam 'i'
                for j=1 to islot
                        if xs(j,i)==0
                        Assign ys number of conflicts of slot 'j'
                        Assign y number of conflicts of slot 'j' after allocating exam 'i'
                        Obtain preference value based on ye and ys
                        Assign xc(j,i) with ye + ys –y + pref
                        end
                end
        end
end
Identify exam with maximum conflict reduction potential
Identify slot to assign 'exam'
update slot conflict xs
allocflag(exam)=slot
```

```
while there is still exams allocated to slot nex+1
        clear y1
        Update 'exam' column of 'xc'
        update the 'slot' row of 'xc'
        for i=1 to number of exams
                if exam i is assign to nex+1
                        j=slot;
                        Assign ye with number of conflicts of exam 'i'
                        if xs(j,i)==0
                                Assign ys with number of conflicts of slot 'j'
                                Assign y the number of conflicts of slot 'j' after
                                allocating exam 'i'
                                Obtain preference value based on ye and ys
                                Assign xc(j,i) with ye + ys – y + pref
                        else
                                Assign xc(islot,i) with 0.3
                        end
                end
        end
        identify exam with maximum conflict reduction potential
        identify slot to assign 'exam'
        update slot conflict xs
        allocflag(exam)=slot;
        if slot==islot
                add additional slot, update xs and xc
        end
end
```

Figure 3.18: Algorithm for Allocation of Exams to Time slots

The above algorithm is divided into three parts, each having a loop to do the allocation of exams to time slots. The first loop is responsible for the first round of allocation, ensuring that the exams with the largest number of conflicts are scheduled first into the slots. The loop has a complexity of $O(n)$ which is proportional to the number of exams. The second loop will schedule exams which have been deferred in the first round of allocation. It is a nested loop with two loops forming the external loop and the internal loop. Both of these loops go through the exams list; thus, giving the element n as the maximum value, which results in a complexity of $O(n^2)$. The final loop is responsible for allocating unallocated

exams which have not been scheduled in the first or second loop. The final loop has a complexity of $O(n^2)$ with the maximum number of time slots to solve the problem is equal to the number of exams, contributed by a for loop nested in a while loop.

Overall, the whole process of allocating exams to time slot has the complexity of $O(n + n^2 + n^2)$, which totals to $O(n + 2n^2)$ and a final complexity of $O(n^2)$.

Effects of Pre-Ordering Exams on Scheduling

In the process of assigning exams to slots, or creating the conflict chain, we have identified that the final outcome is highly dependent on the ordering of the exams prior to the assignment. We can look further into this phenomenon to identify the criteria or reasons for this behaviour. Each exam in the examination scheduling has corresponding exams that clash with it, except for any exam that is taken only by students who are not sitting for any other exam. Whenever there are two students who are both taking the same exam and either of them also has another exam, the clashing situation exists. This situation is depicted in the following figure:

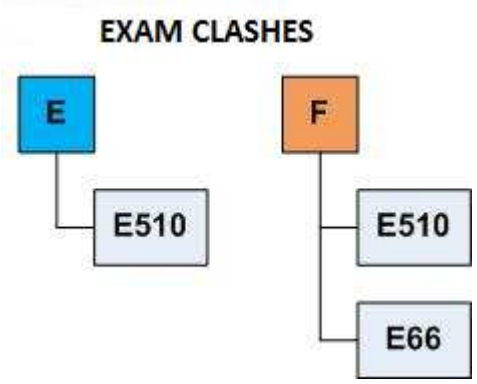


Figure 3.19: Figure Illustrating Exam E510 Clashes with Exam E66

In the above figure students *E* and *F* will both be sitting for exam *E510* and student *F* has an additional exam *E66*. When this situation arises in the examination scheduling problem, we know that *E510* clashes with *E66*; thus, making these two exams interconnected and ensuring that they cannot be scheduled in the same time slot or location in order to adhere to the hard constraint imposed on the scheduling problem. The above instance creates a link of dependence between these two exams. If there exists one exam in a slot then we cannot have its counterpart in the same slot. Another fact that needs to be highlighted is that the two exams *E510* and *E66* actually contributed towards the calculation of the cost function. Whenever these two exams are scheduled less than 5 slots apart, it will add some weight to the cost function.

In an instance where there are other exams that the student is sitting for and between these exams there are other students who are also sitting for it, this would result in an intertwined connection between the exams. This creates a complex interlinking between these exams and determines the outcome of the possible solutions that can be generated during the conflict chains generation, based on the order in which these exams were assigned into slots. To prove this, we introduce a clash list for a set of exams, as depicted in the following figure:

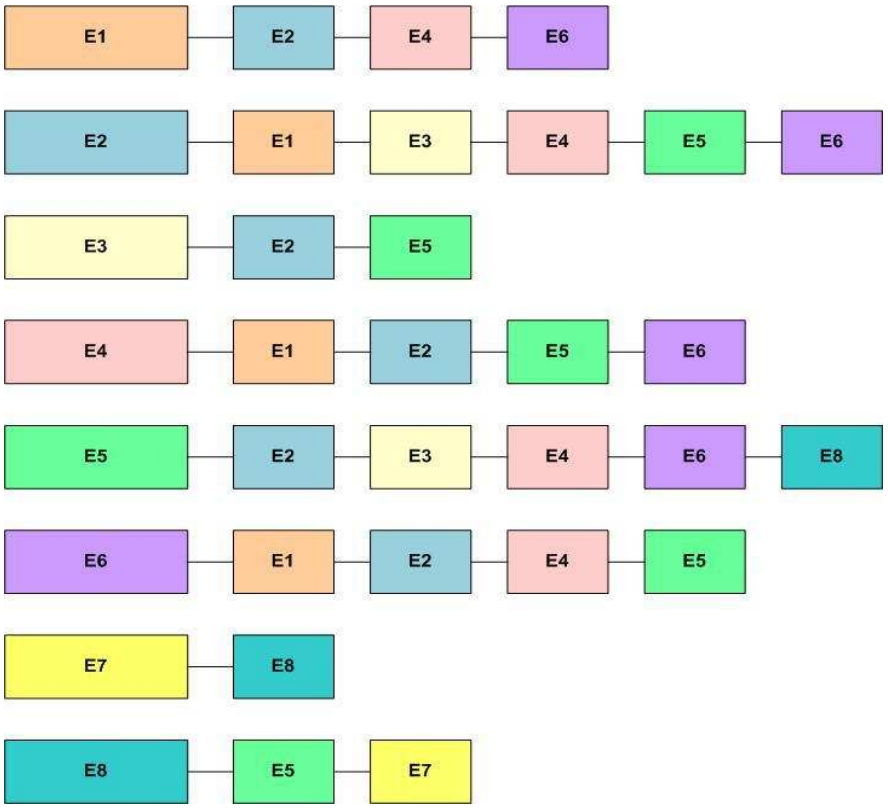


Figure 3.20: Figure Illustrating 8 Exams with The Clash List Pre-ordered Using Ordering 1: Random Ordering (RO)

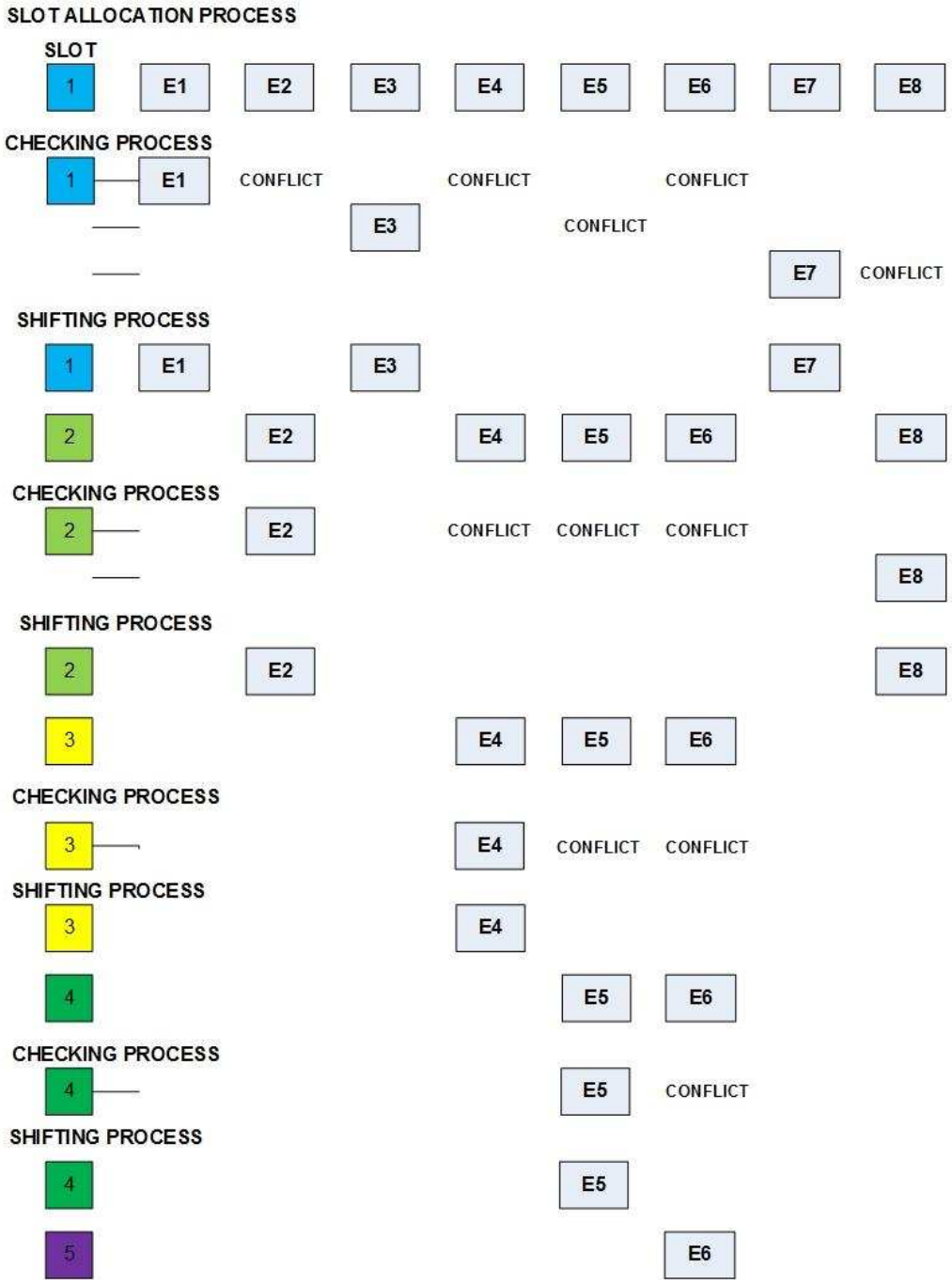


Figure 3.21: Slot Allocation Process for Random Ordering (RO)

In the above and following examples we omit the list of students and other details since the information is no longer needed in the processing. The figure above shows the exam list from *E1* to *E8* (the first column); each is followed by other boxes containing the exam codes for

those exams with which they clash. We have obtained this arrangement for conflict chain creation based on random ordering. The following figure is another ordering of the same datasets which we have obtained through the Largest Degree arrangement.

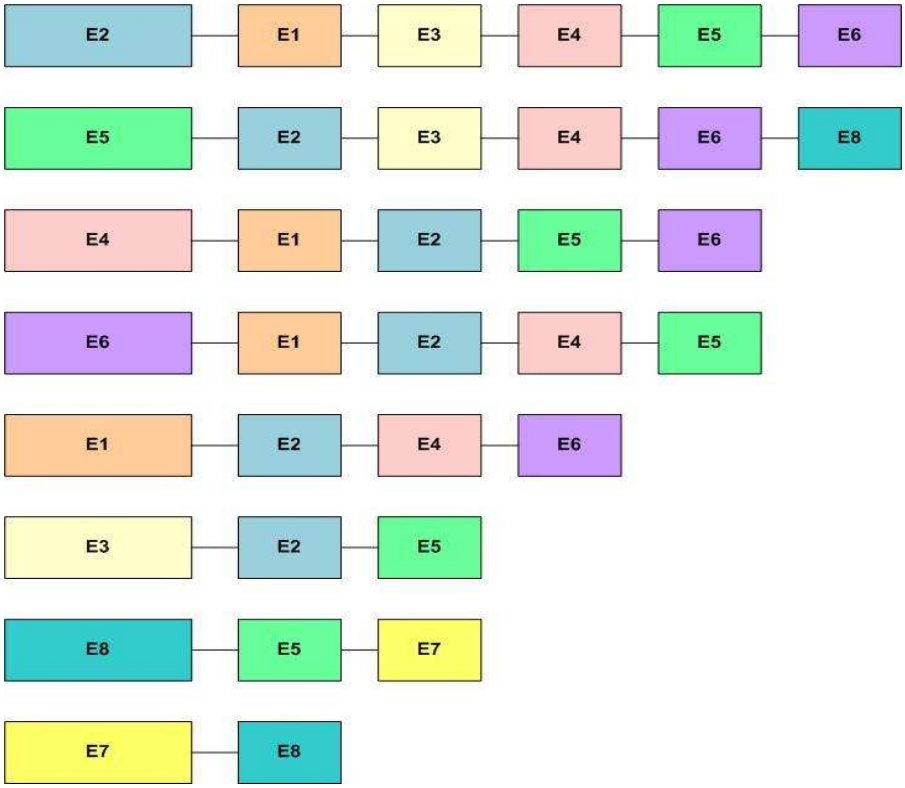


Figure 3.22: Figure Illustrating 8 Exams with The Clash List Pre-ordered Using Ordering 2: Largest Degree (LD)

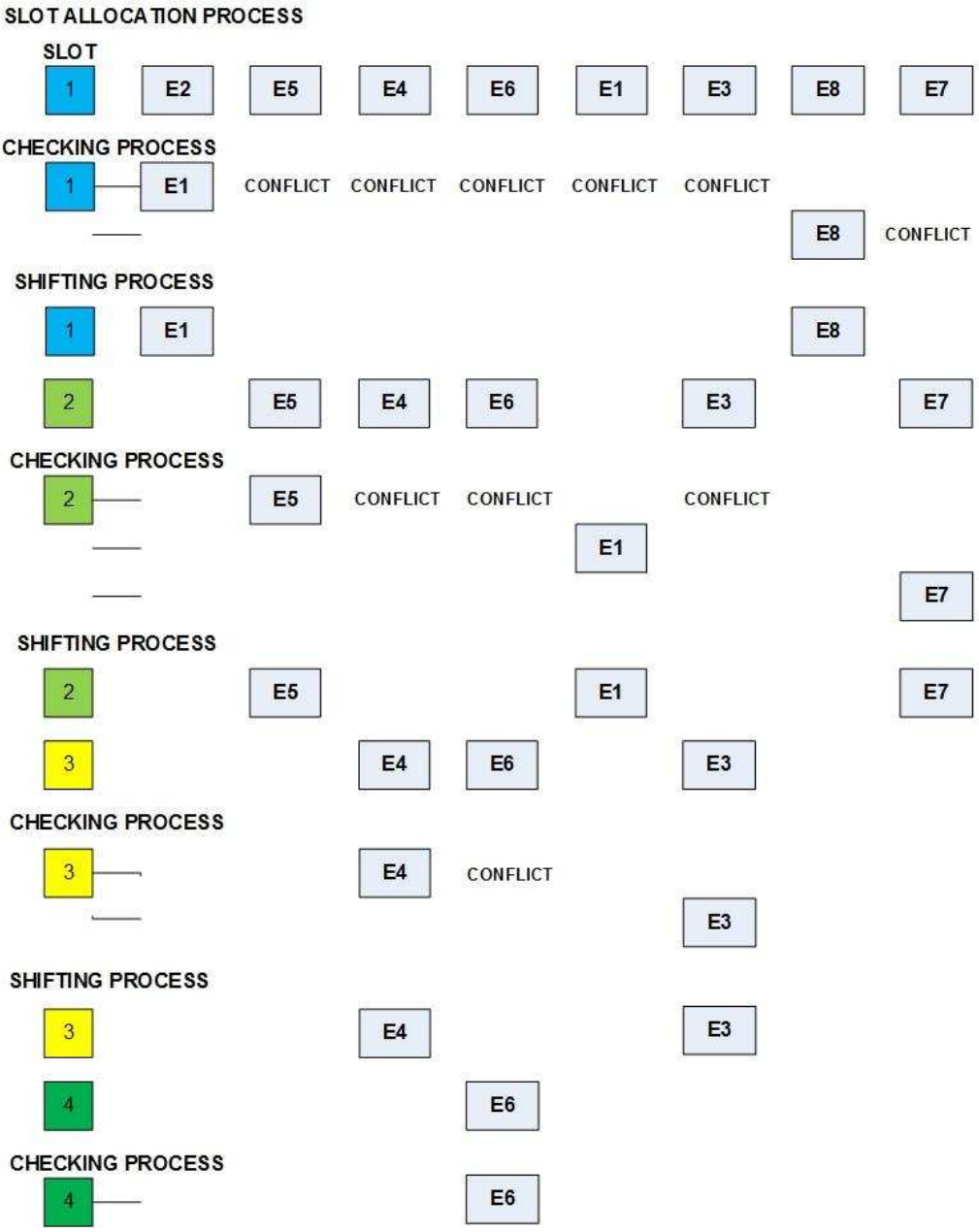


Figure 3.23: Slot Allocation Process for Largest Degree (LD)

Figure 3.20 and Figure 3.22 are translated into two matrices as shown below: *clashA* and *clashB,* respectively. Pre-processing has been achieved by running the code to determine the number of minimum slots required. The slot allocation process for the Random Ordering is shown in Figure 3.21 and slot allocation process for Largest Degree is shown in

Figure 3.23. As a result of going through the slot allocation process, the following number of slots required for each ordering is obtained:

clashA=[
1 2 4 6 0 0
2 1 3 4 5 6
3 2 5 0 0 0
4 1 2 5 6 0
5 2 3 4 6 8
6 1 2 4 5 0
7 8 0 0 0 0
8 5 7 0 0 0];

clashB=[
1 2 3 4 5 6
2 1 3 4 6 7
3 1 2 4 5 0
4 1 2 3 5 0
5 1 3 4 0 0
6 1 2 0 0 0
7 2 8 0 0 0
8 7 0 0 0 0];

a) Ordering 1 *(clashA)*: 5 slots

b) Ordering 2 *(clashB)*: 4 slots

We have also done some pre-processing on the problems of benchmark datasets to determine the minimum number of slots required to schedule the exams and, as expected, different orderings have produced different results. The differences can be seen in Table 3-1:

Table 3-1: Different Number of Slots Generated After Pre-Processing By Using Different Pre-Orderings

Name of Dataset	Minimum No. of Slots Required Using Random Ordering (RO)	Minimum No. of Slots Required Using Largest Enrolment (LE)	Minimum No. of Slots Required Using Largest Degree (LD)
nott	26	19	18
car-s-91 (I)	44	35	32
car-f-92 (I)	48	36	34
ear-f-83 (I)	29	26	24
hec-s-92 (I)	22	22	20
kfu-s-93	25	21	20
lse-f-91	22	20	19
pur-s-93 (I)	54	41	38
rye-f-92	28	26	25
sta-f-83 (I)	35	35	35
tre-s-92	29	22	23
uta-s-92 (I)	43	37	36
ute-s-92	13	10	11
yor-f-83 (I)	29	25	27

Based on these results we can generalize that different pre-orderings result in a different number of slots being required and this will also affect the quality of the schedules later on.

Implementations of Backtracking to Reduce the Number of Slots

Reducing the number of slots for a solution reduces the number of days and resources that will be utilized for the examination, thus reducing the operational cost. However, by reducing the number of days, it will

definitely increase the value of the cost function since the Carter cost (2.1) function is highly dependent on the temporal distance between consecutive exams, which is affected by the number of days' duration of the overall examinations.

During the scheduling process, the order of processing the exams may sometimes lead to a non-optimal assignment of exams to slots which could create an infeasible schedule (i.e. does not satisfy the number of slots required). This situation calls for a reassignment of exams from the initial slot allocation to other slots in order to ensure the number of slots is reduced to the required number and the schedule becomes feasible. This kind of reassignment will need to revisit or backtrack through the initial allocation or assignment process, and therefore we will call this a backtracking process. In the backtracking process, some assignments which have already been made will be undone in order to schedule these exams in other time slots. As a result, this simulates the generation of a set of feasible schedules that will be used in the optimization process later.

The backtracking process takes place when we execute the optimization stage to minimize the number of slots for a solution. The main objective of the algorithm is to look for possible exam movements within the available slots and identify the best moves that can be made.

The specific objectives of the backtracking might include: 1) to reduce the number of slots in order to satisfy the slots number requirement in a given problem; 2) to prepare the non-optimal schedule for further optimization; or 3) to undo certain assignments of exams to periods during scheduling in order to allow other exams, which previously failed to

be assigned and caused the infeasibility of the schedule, to be scheduled first.

In one of their approaches, Carter *et al.* (1996) utilized the backtracking process in the main algorithm to come up with a feasible solution for a timetable, giving the algorithm the advantage of undoing steps; which is de-assigning exams from a period to obtain a previous solution state, with the objective of assigning an exam which previously could not be assigned to any one of the periods or slots. Carter *et al.* (1996) concluded that the backtracking process managed to reduce the overall solution length by 50%; thus, we found the algorithm very appealing and it fitted easily into our implementation. Therefore, it was decided to use the Carter *et al.* (1996) backtracking algorithm, with some modifications, as the basis from which to eliminate or reduce the slots of the current solution.

This is due to the fact that a reduction of slots involves rearranging or reassigning allocated exams to new slots, which will result in the modification of other related exams. By doing this, we are in the same position as Carter *et al.* (1996), as the probability of the future movement of exams to reach a feasible solution is uncertain; thus, we need to have the capability to undo any movements made previously.

This is in anticipation of the fact that by reducing the number of slots at the early stage, one can minimize the cost of timetables at the later stage during the optimization process. The initial schedule with a few slots (i.e. less than the required number of slots) can always be modified to one with the required number of slots. We hypothesize that this could provide

a useful buffering space during the optimization involving permutations of exam slots. Consequently, this has the potential to improve the quality of the schedules (Rahim *et al.*, 2009; Rahim *et al.*, 2012).

After each exam has been assigned to a slot via the scheduling process, backtracking is then performed to further reduce the number of slots, if any reduction is possible (Rahim *et al.,* 2013b). The backtracking process in our proposed framework is illustrated by the following diagram.

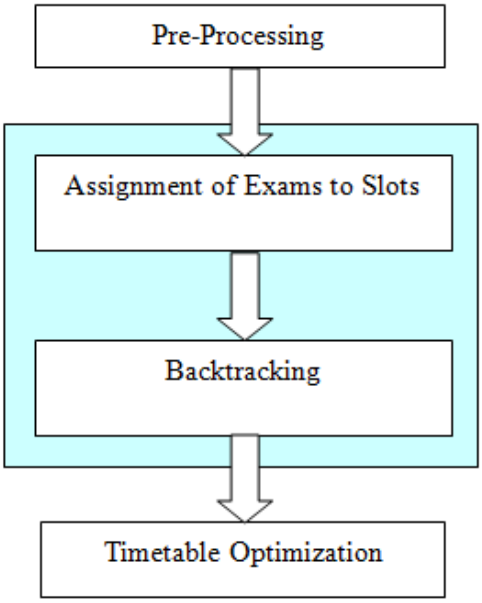


Figure 3.24: Backtracking Stage in Our Proposed Framework

We have implemented the backtracking process used by Carter *et al.* (1996). The backtracking took place after doing the scheduling using the allocation method, as discussed in the previous section. The purpose of implementing this is to see whether backtracking could reduce the number of time slots required to schedule the exams.

The flowchart of the backtracking process implemented in our work is given in Figure 3.25. Note that the general idea was based on the

backtracking algorithm proposed by Carter *et al.* (1996), but with some modifications to suit our framework. Figure 3.26 outlines the pseudocode of the whole process.

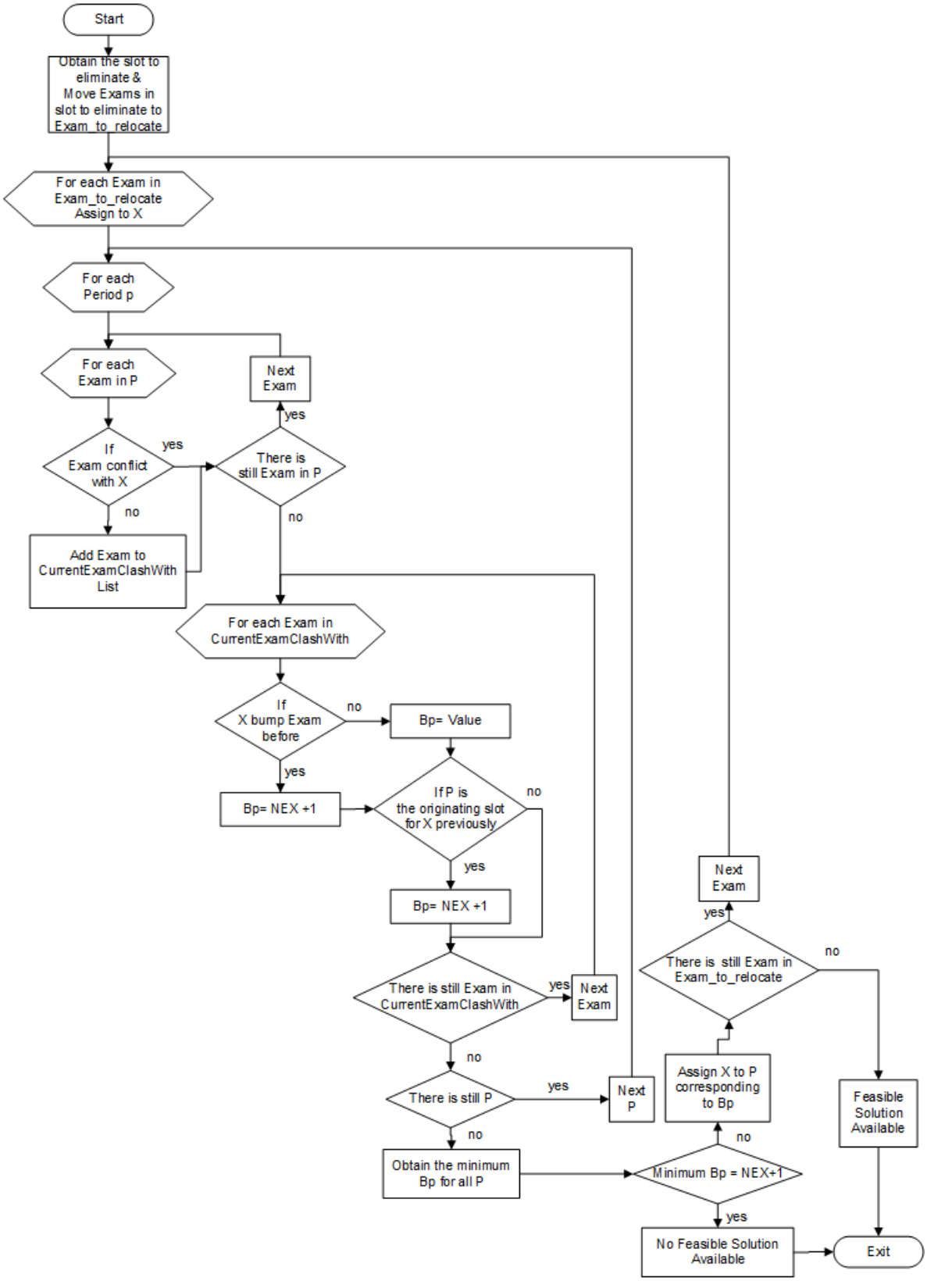


Figure 3.25: Flowchart of Backtracking Process

Algorithm 4

```
for i = 0 to Last_i
        for p=0 to Last_p
                if <i can fit to p>
                        Assign i to p

                end if
        next p
        for p = 0 to Last_p
                Bp = 0
                x = all course where i conflicts
                for j = 0 to max(x)
                        APaper = x[j]
                        for p = 0 to Last_p
                                if <APaper can fit to p>
                                        CostApaperAtP = <cost if Apaper is put
to p>
                                end if
                        next p
                        if <total CostP = 0>
                                if <i bumped APaper before>
                                        Bp = -1
                                else
                                        Bp = Bp + 1
                                        Mark APaper to bump if p selected
                                end if

                        else
                                <Get min cost and mark p for new location of
APaper>
                        end if
                next j
                Calculate m for p
        next p
        Get min Bp
        if <min Bp = 0>
                Get all of p where Bp= 0
                Get min Mp for all p selected
                put i to p
                execute APaper shifting
        else
                if min Bp is infinity
                        mark i as unable to schedule
                else
                        Get all p for min Bp
                        if p unique
                                put i to p
                                execute APaper (which is marked) shifting
                        else
```

```
                                get min Mp for p
                                put i to p
                                execute APaper (which is marked) shifting
                        end if
                end if
        end if
Next i
```

Figure 3.26: Pseudocode for Backtracking

For each exam in the *Exam_to_relocate* list that we have selected (which will be referred to as the current exam), we will calculate a *Bp* value for each slot that we have as the solution, which is the number of exams that will need to be relocated if the current exam is assigned to the slot *p* (the process of finding the number of exams clashing with it in each of the available periods). Please note that the exams clashing with the current exam will be bumped to the *Exam_to_relocate* list, and thus will be assumed to be unscheduled exams.

In the process of calculating the *Bp* value for each slot we create a *CurrentExamClashWith* list for each slot that the exam can enter or be relocated. All the exams which have students clashing will be included in this list. The total number of the content of *CurrentExamClashWith* is the *Bp* for the slot. If the exam is allocated there the content of *CurrentExamClashWith* can easily be used to populate the *Exam_to_relocate* list.

Initially, each *Bp* value in each slot is assigned the value of 0 and if an exam cannot be assigned to the slot for a specific reason, it will be marked or given the value *number_of_exams + 1*. In the process of finding the *Bp*, if the exam in the list has bumped any clashing exams encountered

in the period we are dealing with, then the *Bp* for this period is equal to *number of exams + 1 (Bp= nex + 1).* This is a bit different from Carter *et al.* (1996), who assigned an infinity value to the *Bp* whenever they encountered this condition. We also assign *Bp = nex+1* for a period, if the exam in the list originated from this period. This is another modification to Carter's method to avoid a cyclic shift. We continue finding the *Bp* for all periods for each exam in the waiting list.

Each of the exams that can be relocated to accommodate the coming exam has an indicator to determine whether the incoming exam has a history of shifting out the exam to the relocation list. We create a *BumpMatric* which is a matrix of *exam* x *exam*, where the rows represent the *Exam_to_reduce* and columns represent *CurrentExamClashWith*. The intersection between rows and columns has an indicator: the value 1 indicated the *Exam_to_reduce* has bumped the corresponding exam *CurrentExamClashWith*. The value 0 indicates that *Exam_to_reduce* has not bumped the corresponding exam *CurrentExamClashWith.* This value, however, will change to 1 in the transfer stage if a 'bump' occurs.

We have taken the same approach as Carter *et al.* (1996) in that an exam is allowed to push out an exam to the relocation list only once during the process; this is to eliminate the probability of creating a cyclic shift resulting in an infinite loop of transferring exams out and into the slot. In order to do this, we monitor or keep track of the last slot that an exam in the relocated list originated from. This is to ensure that the exam that has been transferred out does not go back into its original slot when it is time for the exam to be evaluated for relocation.

The purpose of finding the *Bps* for all the periods is to determine which period to choose to assign the exams in the waiting list. Bps for all periods can range from the value of 0 to *nex + 1*. So, the best *Bp* would be 0 and the worst *Bp* would be *nex + 1*. This means that the exam in the waiting list will be assigned to the period with the minimum value of *Bp*. The lowest *Bp* value will be the best criterion to be selected as the target location for the *Current* exam relocation.

In the period selection stage, there is always a possibility of having the same *Bp* values. If there are several periods with *Bp* = 0, then our method will choose the first period with *Bp*=0 encountered or, in other words, the first available period with no exams clashing with the exam in the waiting list. In cases where the *Bp* ranges from the value *1 to nex (Bp=1 to Bp=nex),* and there exist multiple periods with the same *Bps*, then our method will execute a selection based on the weighting given to the periods.

The weighting given was based on the total number of students having conflicts in both exams in the periods and the exam in the waiting list. The period with the maximum value of the weighting will be selected; thus, the exams in the period clashing with the exam in the waiting list will be bumped to the waiting list. The weighting given is mainly for the purpose of breaking the ties of the same *Bps*.

Once the period or the location to assign the exam in the waiting list is complete, the transfer stage follows. The transfer stage is the process of transferring the current exam in the waiting list to the new period selected.

The above process then repeats for other exams in the waiting list. If, at the end of the process, some exams fail to be assigned to any periods, then we assume the backtracking process has failed; thus, the above process will be undone and the previous configurations of allocation of exams to periods will be used. The transfer stage will allocate the exam to its new slot and it will also transfer out existing exams in the slot that clashed with the incoming exam to the *Exam_to_relocate* list. All current information that is affected by the move is initialized to its original value before starting the evaluation for the next exam in the *Exam_to_relocate* list. If the algorithm finds a situation where there is no solution to allocate all the existing exams or any of the exams in the *Exam_to_relocate* list, then it will revert and undo all of the movements of exams to obtain the original placement before the reduction of the slot is executed.

The backtracking algorithm consists of a few levels of nested loops that will increase the computational complexity. This is due to the fact that we will be traversing and searching the solution space for all possible moves that an exam can make and all moves are evaluated. The first loop will traverse the list of exams that have to be selected to be relocated. Within this loop there are two sequential loops. The first will traverse all the available remaining slots to check if the exam can be allocated to the slot; and, if this is possible, an allocation of the exam to the slot will take place. The second loop will go through all the available periods and evaluate the possibility of assigning the exam to other slots which have conflicting exams. Within the second loop there are two loops; one inside the other. Each of these loops has a different controlling logic. The complexity of the main loop depends on the number of exams that need

rescheduling and would have a maximum value of n, being the number of exams. The two sequential loops inside the main loop are controlled by the number of slots currently available and required by the solution m. The algorithm initially will have the complexity of $O(n(m+m(n(m))))$. As the value of m and n grow bigger, the value of m will be the same as n. The initial algorithm complexity reduces to $O(n(n+n(n(n)))) = O(n(n+n^3))$. This can be further reduced to $O(n^2 + n^3) = O(n^3)$.

Types of Backtracking Implemented in the Proposed Framework

i. First Method: Backtracking 1 (BT1)

In the first backtracking method, called here Backtracking 1 (BT1), we attempt to eliminate the last utilized time-slot. We have implemented the backtracking process used by Carter *et al.* (1996), but with some modifications. In contrast to Carter *et al.* (1996)'s method, where backtracking was performed during the initial placement of exams, in our approach the placement of exams to their allocated slots has already been completed; therefore, we are attempting to convert the infeasible schedule into a feasible one.

After allocations of exams to slots were completed, we identified all the exams in the last slot and we assigned them to a waiting list of unscheduled exams. Then, for each exam in this list, we initialized the selection criterion, which is known as *Bp* (according to Carter *et al.*, 1996), for all periods equal to zero *(Bp=0)*. Next, for each exam in the list we

proceed by finding the number of exams clashing with it in each of the available periods. *Bp* for each period is the number of exams clashing with the exam currently being evaluated in the waiting list. Please note that the exams clashing with the exam in the list are the exams that will be bumped to the waiting list, and thus will be assumed to be unscheduled exams. (Note also that we process the exams in the list on a 'First In, First Out' basis).

ii. Second Method: Backtracking 2 (BT2)

In the second backtracking approach (BT2), the objective is to eliminate the slot containing the fewest number of exams after the allocation method. The number of slots that will be eliminated is also 1 (the same as BT1).

It is interesting to note here that, in BT2, the slot that will be eliminated could be any slot in the schedule (in BT1 it is always the last slot); therefore, it could be the first, in the middle or the last one. Once the slot with the fewest exams has been determined, all the exams will be put in a waiting list. Each exam in the list will be evaluated for reallocation as with our first approach (BT1).

Differences between Carter's Backtracking and the Proposed Backtracking

The backtracking implemented by Carter *et al.* (1996) was used during the initial placement process. However, in our approach the placement of exams in their allocated slots has been completed. What we are doing is

using the backtracking method to rearrange the placement of exams to reduce the final number of time slots to schedule all the exams. We differ in terms of approach and purpose from the backtracking of Carter *et al.* (1996); we are utilizing the backtracking process to reduce the number of slots of an existing feasible solution and the other is utilizing the process to allocate exams which could not be allocated via the normal process. Thus, two different outcomes will be derived from the process, as depicted in the two following figures.

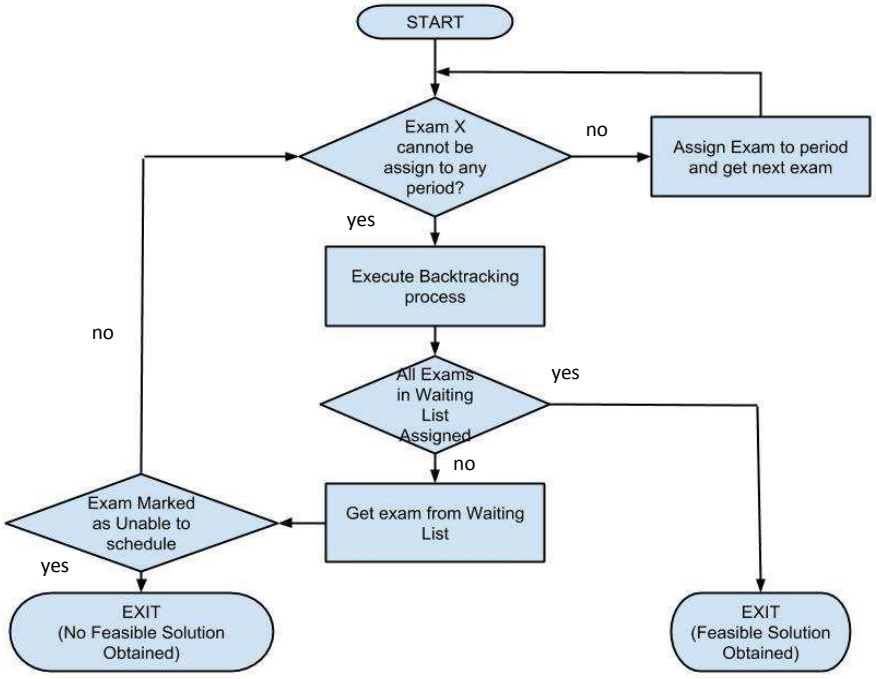


Figure 3.27: Flowchart for Carter's Backtracking in General

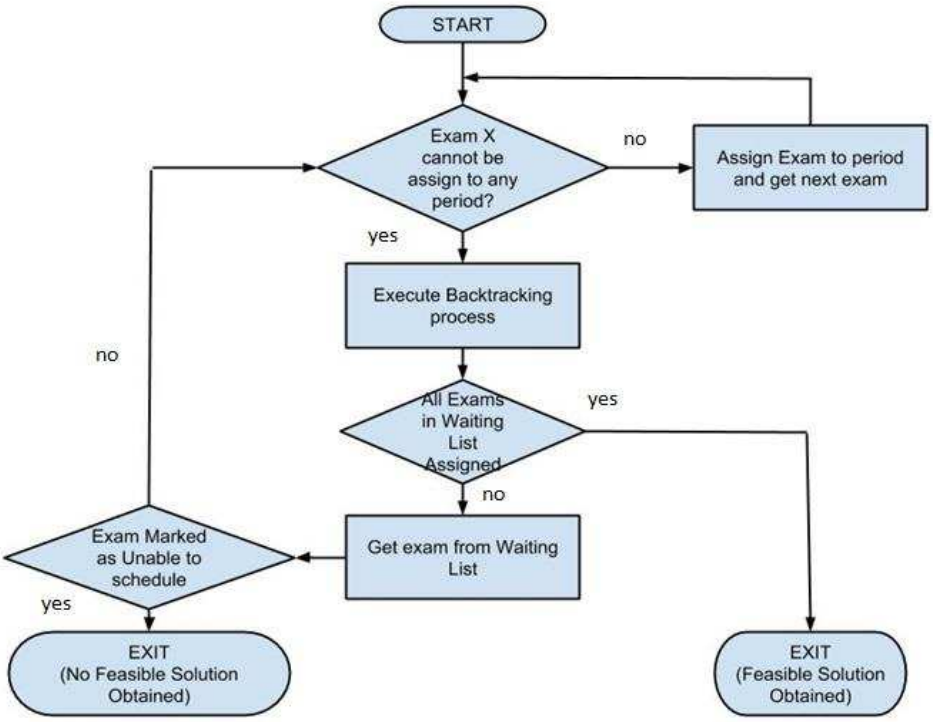


Figure 3.28: Flowchart for our Backtracking in General

In our approach, after the allocation of exams to slots is completed, we identify all the exams in the slot to be eliminated and we assign them to a waiting list which is a list of unscheduled exams. Then, for each exam in this list, we start to calculate and evaluate the possible locations to be assigned. Only when the assignments have been made are exams which are in conflict with the incoming exam transferred out to the unscheduled list. Carter *et al.* (1996), on the other hand, transferred out the exams which were in conflict with the incoming exam straight to their new slots and only exams that could not be allocated to other slots were shifted to the unscheduled list or waiting list.

We have created and converted Carter *et al*. (1996)'s backtracking algorithm to a flowchart for comparison and to give a better understanding of the process, as can be seen in Figure 3.29.

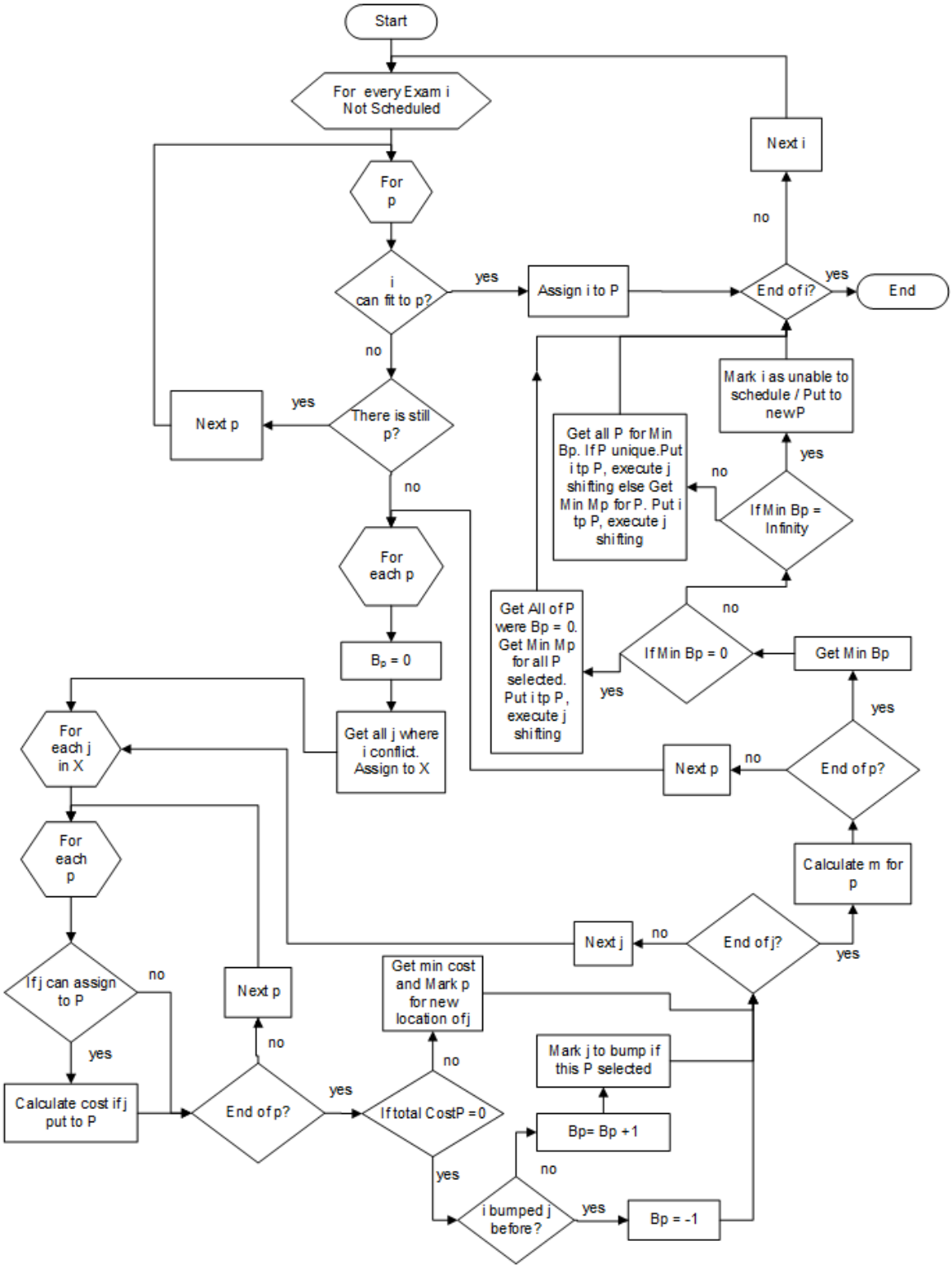


Figure 3.29: Flowchart for Carter *et al.* (1996)'s Backtracking in Detail

A scheduling process which utilizes an allocation method to assign each exam to a slot using a Graph Colouring Heuristic, coupled with a backtracking procedure (a modified version of Carter et al. (1996)'s backtracking approach), is adopted as a basic scheduling process. It is expected to produce ***only feasible solutions*** *with a total number of slots that will satisfy the slots number requirements given in the problem. Backtracking will aid in assuring the feasibility of the schedule by reducing the number of slots if the allocation method does not conform to the constraint on the number of slots. Besides ensuring the feasibility, by reducing the number of slots at the early stage, the extra slots could provide a useful* ***buffering space for subsequent optimization*** *in improving the quality of the schedules.*

3.2.4 Optimization

In the area of computer science or mathematical programming, optimization can be understood as selecting the best solution from a set of available solutions. In general, optimization can be seen as a process that maximizes the benefits while minimizing the investment in resources that facilitates these benefits.

Therefore, in the examination scheduling context, optimization could be defined as a process of improving the quality of the feasible exam schedule or solution. A feasible timetable could have an ordering of exams that does not satisfy many of the soft constraints. This calls for a separate

deployment of optimization to achieve greater satisfaction of soft constraints and consequently the improvement of the quality of the schedules.

In order to demonstrate optimization as a process to improve the examination schedules, we present below a diagram (Figure 3.30) which illustrates an example of a feasible examination schedule. As can be seen, there are a few students: *S1, S2, S3, S4 ,S5 …. Sn* and a few exams*: E1, E2, E3, E4 …Em* together with a few time slots: *T1, T2, T3, T4 …. Tk*.

In this example, student *S1* has registered for exams *E1*, *E2* and *E4*; and student *S2* has registered for *E3* and *E4*. Therefore, exams *E1*, *E2* and *E4* are the set of conflicting exams for student *S1* and because of this, they cannot be assigned to the same time slots. The diagram below shows that these three exams are not assigned to the same time slot (they are assigned to time slots *T1*, *T2* and *T3* respectively) and thus this is considered a feasible examination schedule.

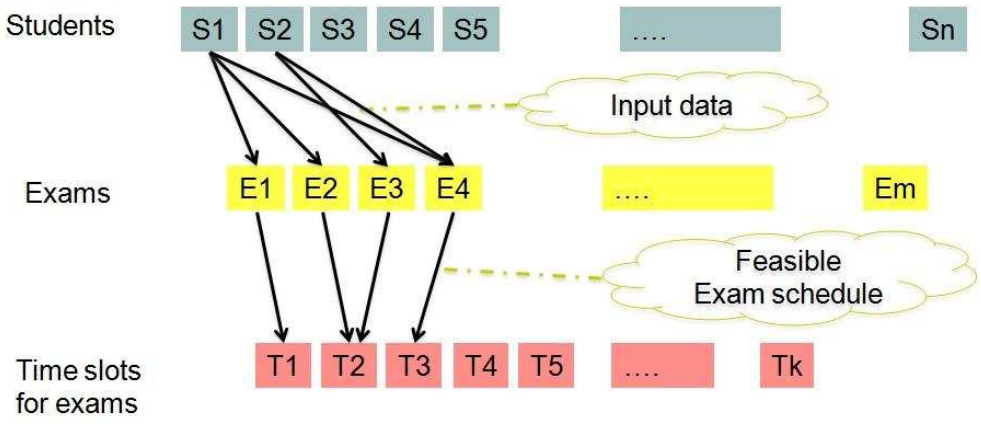


Figure 3.30: An Example of a Feasible Examination Schedule

According to the above example, although student *S2* has a feasible examination schedule, the timetable does not satisfy the soft constraint in terms of putting a gap between one exam and the next exam that student will have to sit. This is not necessary but satisfying this would improve the schedule quality by benefiting the student, since it allows the student to have more revision time between exams. Thus, if exam *E3* is now reassigned to time slot *T1,* the quality of the schedule can be improved, as illustrated in Figure 3.31.

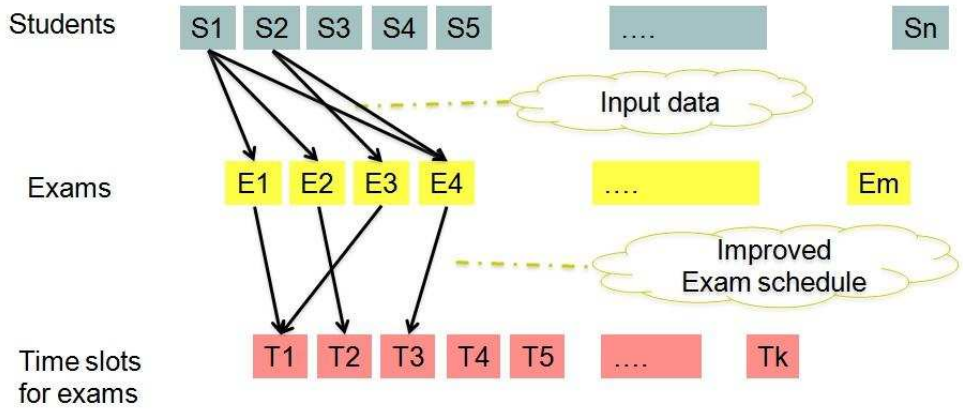


Figure 3.31: An Example of an Improved Examination Schedule

This study has adopted an approach to the design of the exam schedule optimization that focuses on promoting understandability of the optimization process. To this end, we have avoided random exploration of the solution space, such as that widely proposed in the literature, where mostly were not really successful in applying to a wider range of timetabling problems, and this scenario calls for an investigation on raising the generality of the existing approaches (Qu *et al.*, 2009a). Thus we have adhered to the deterministic evaluation of the search direction

during the optimization process, which through systematic procedures, by using the background knowledge about the structure of the cost function, the optimization process will always maneuver intelligently to achieve as low cost as possible (rather than randomly search for solutions).

In this work, we develop optimization methods to improve the initial feasible schedule generated by the allocation method. The cost of the initial feasible schedule is normally fairly high. In order to minimize the cost, we perform the minimization of the total slot conflicts, followed by further optimization of the initial schedule by the permutations of exam slots and the reassignment of exams between slots (Rahim *et al.,* 2012).

The standard objective function that we will be using is the cost function originally proposed by Carter *et al.* (1996), as discussed in Chapter 2. The lower the cost obtained, the higher the quality of the schedule, since a gap between two consecutive exams allows students to have extra revision time.

3.2.4.1 Minimization of Total Slot Conflicts

The notion of a slot conflict is a generalization of the notion of exam conflict. We consider two exams i and j as being "in conflict" if there is a student who is taking both exams. In a feasible schedule such exams are allocated to different exam slots. It is worth noting here that the conflict between exams is a binary property that does not increase in value if there are several students taking these two exams. Consequently, once we establish which exams are in conflict we do not need to be distracted, in the exam scheduling process, by the detailed student-exam data. This

domain-transformation approach, motivated by the granular information processing paradigm (Bargiela and Pedrycz, 2002) provides a key advantage of the proposed exam scheduling.

Taking an even a broader view on the exam conflict, a novel contribution of this study is the consideration of the exam-slot conflict. Since every exam that is in conflict with the exam *i* is allocated to other slot in the initial feasible solution, we can count the number of slots that contain conflicting exams for exam *i*. An exam-slot conflict is a matrix (binary matrix) with a dimension of *number of exams* x *number of slots*. The value 1 in the matrix at location *i,j* indicates an exam *i* has a one or more conflicting exams in slot *j*.

To exemplify the exam-slot conflict matrix, assume that we have an exam *E1* that is in conflict with *E3* and *E7*. After the initial allocation method, let us say, *E1* is assigned to timeslot *T2*, *E3* is assigned to time slot *T7* and *E7* is assigned to time slot *T10*. The exam-slot conflict matrix [*E2, T7*], [*E2, T10*] will have the value 1 and the total number of exam-slot conflicts for *E1* is 2 which was contributed by *T10* and *T10* (which is the total number of columns in the matrix having the value of 1 for a particular exam). The best case scenario is an exam that is not in conflict with any exam in other slots, having a total number of exam-slot conflicts value of 0. On the other hand, the worst case scenario is an exam having a total number of exam-slot conflicts of *number of slots -1* (the exam has one or more conflicting exams in all other slots.)

In order to guide our exam schedule optimization process, we use the total number of exam-slot conflicts as a measure of the ability to re-schedule exams between the slots. If the total count is high, it means that, on average, exams are in conflict with many slots and consequently there are few slots available for rescheduling. Conversely, if the total count is low, on average, there are more slots that can be used for the re-scheduling of exams. To the best of our knowledge, the potential for the rescheduling of exams has not been quantified in the literature so far, despite it being a key factor in enabling the improvement of the initial feasible schedule.

*Consideration of the **exam-slot conflict** in the optimization is a novel contribution of this study. Slots that contain conflicting exams can be counted easily. The exam-slot conflict matrix has a binary property that indicates an exam conflicting slot. The exam-slot conflict value does not increase if an exam has several conflicting exams in a particular slot (and therefore is equal to 1). But the total exam-slot conflict for a particular exam which has conflicting exams allocated to a few different slots will be correspondingly higher (equal to the number of slots containing the conflicting exams). A high total of exam-slot conflicts indicates that, on average, exams are in conflict with many slots and consequently there are few slots available for rescheduling, and vice versa. As far as the potential for rescheduling is concerned, to the best of our knowledge, it has not been quantified in the literature although it acts as a key factor in enabling the improvement of the initial feasible schedule.*

Recognizing the rationale for the maximization of the ability to reschedule individual exams between different slots, we start our optimization process by minimizing the total exam-slot conflict.

The procedure starts by taking the first exam *i* in the dataset, and calculating the total number of slot conflicts. Next, we try to reassign exam *i* to all other valid slots (i.e. not in conflict with exam *i*) and calculate the new total of slot conflicts. A slot that could lead to a maximum reduction of total conflicts will be selected as the new slot for exam *i*. The procedure is repeated for all other exams in the problem. The pseudocode for minimizing slot conflicts is presented in Figure 3.32.

By minimizing the total number of slot conflicts it is usually possible to reduce the cost of the exam schedule. However, we consider this stage primarily as the enhancement of the potential for the subsequent minimization of the cost of the schedule.

Nevertheless, it is worth observing that although the cost formula (2.1) counts the spread of exams from the viewpoint of individual students, it is an integrative measure that is concerned with the average inter-exam spread. By reducing the total exam-slot conflict, we achieve a greater packing of conflicting exams and, by implication, an increased possibility of separating the slots that have the largest number of conflicting exams.

Algorithm 5

```
For each exam i in the problem
    Obtain the slot number (where it is allocated) from the allocation flag
    Find the sum of the total slot conflicts, and set it as the lowest total slot
    conflicts
    For each slot (except the slot for exam i)
        Calculate the new total slot conflicts by reassigning exam i into a new slot
        If the new total slot conflicts is lower than the lowest total slot conflicts
```

```
                Set the new total slot conflicts as the lowest total slot conflicts
            End
        Reassign exam i to the slot that produced the lowest total slot conflicts
        End
End
```

Figure 3.32: Algorithm for Minimization of Total Slot Conflicts

Minimization of total slot conflicts involves shifting or swapping exams between slots to find the best location/slot for an exam arrangement that will generate the lowest penalty; thus, reducing the Carter cost (2.1) and providing a better solution. The first initial loop is determined by the number of exams, n. The second inner loop traverses through all of the available slots, m. This gives an increased complexity of $O(mn)$. $O(mn)$ *is equal to* $O(n^2)$ if the number of slots matches the number of exams i.e. one exam in a day. However, there is a limiting factor in the number of m. Regardless of the situation, if the number of exams is to increase, the number of m is limited to the number of days (having exams), thus minimizing the computational complexity to $O(n)$.

Minimization of the total exam-slot conflict *is the first optimization stage proposed in this study. In this procedure, we simulate the reassignment of each exam in the problem to all other valid slots and calculate the new total slot conflicts. The slot that could give the biggest reduction of conflicts will be selected for the reassignment. By minimizing the total slot conflicts it has a huge possibility of reducing the cost of the schedule. However, we consider this stage primarily as the enhancement of the potential for the subsequent minimization of the cost of the schedule.*

3.2.4.2 Minimization of Costs via Permutations of Exam Slots

The second stage of the optimization is explicitly focused on the minimization of the cost function (2.1). The preparatory work of preparing the exam spread data structure, coupled with the maximization of the possibility of re-positioning (re-labelling) exam slots, brings dividends in terms of having a much smaller slot-optimization problem to consider while capturing the essence of the overall exam scheduling problem. Since the number of available exam slots is typically quite small, the optimization of the position of individual slots can be accomplished by the permutation of rows/columns of the spread matrix and the evaluation of the resulting cost (2.1).

Figure 3.33 below shows how the permutation of exam slots has changed the original ordering of the slots in Figure 3.31, and consequently an improved schedule has been generated. By this permutation, a time slot has been added between time slot *T2* and *T3,* and thus giving extra time for the students to do their revision.

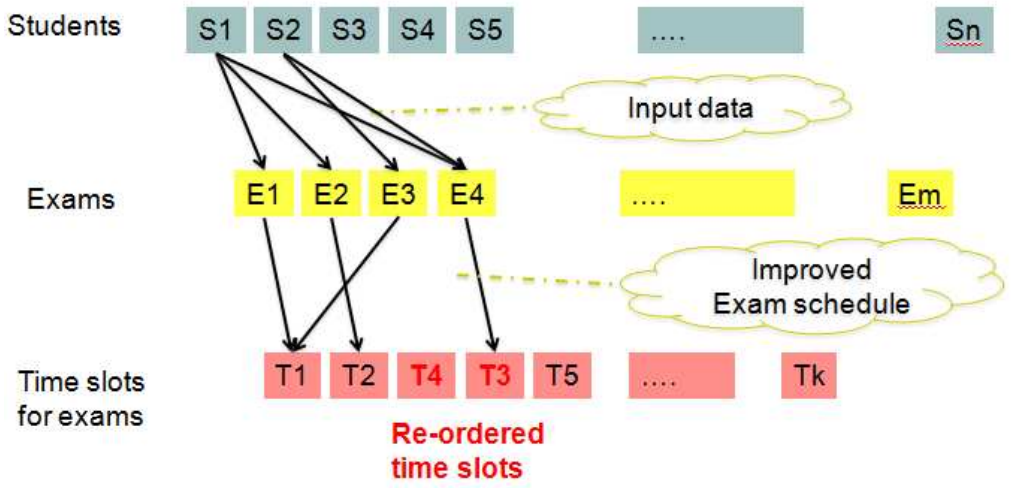


Figure 3.33: An Improved Examination Schedule after Optimization (Permutations of Slots)

However, adding an extra time slot between *T1* and *T2* will have a greater effect as illustrated in Figure 3.34 than adding it between time slot *T2* and *T3* as illustrated in the previous diagram. In this new example, both students have more time between exams as compared to the previous example.

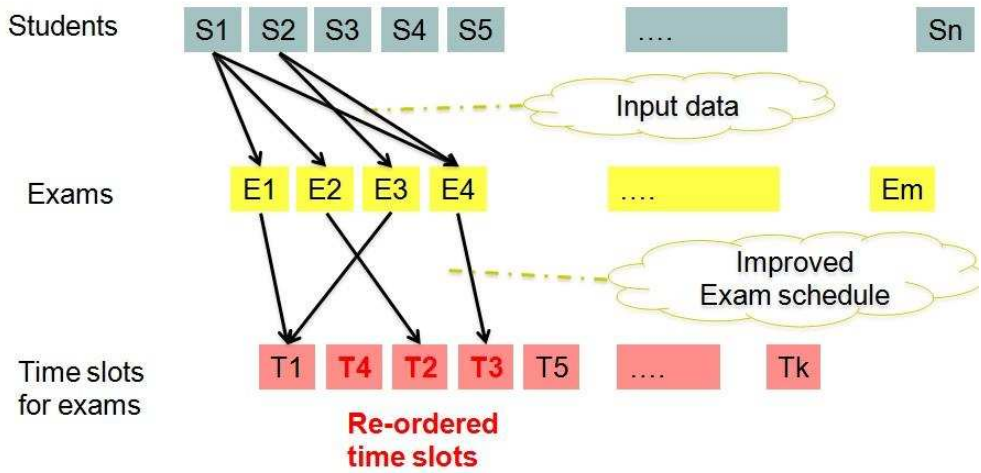


Figure 3.34: Re-ordered Time Slots Via Permutations of Slots with Greater Effect

We propose here three strategies for optimization of the exam spread, named as Method 1, Method 2 (Rahim *et al.,* 2009) and Greedy Hill Climbing (Rahim *et al.,* 2012; Rahim *et al.,* 2013a). A brief explanation of each method is given below:

Method 1

The first method is focused on extracting the smallest element in each row of the original spread matrix and re-numbering the relevant time slots in order to place the smallest element on the first minor diagonal. While implementing such re-numbering, it is possible that higher order minor diagonals will have some greater elements associated with them. However, we suspect that if the primary concern is to minimize the number of adjacent exams, the method provides the optimum solution.

Method 2

The second method takes a different approach of identifying the smallest elements in both rows and columns and shifting them towards the first minor diagonal. This corresponds to arranging simultaneously from the first slot and from the last slot towards the middle one. We believe by taking this approach a more balanced re-numbering is achieved that attempts to minimize the sum of higher minor diagonals.

Greedy Hill Climbing

Based on the ideas from the two methods presented above, we can see the potential of shuffling the exam slots in the spread matrix in order to reduce the cost of the schedule. Here we present another prospective optimization process, by doing the permutations of exam slots in the spread matrix. This process involves the shuffling of slots or columns as block shifting and swapping. The procedure started by reading a spread matrix which is a matrix indicating how many students are taking an exam from slot *'i'* and *'j'*.

The permutations in the spread matrix involved the swapping of slots and repetitions of block shifts. Each slot will be swapped with another slot. This is done by doing provisional swapping and the Carter cost (2.1) will be evaluated first. If the cost is reduced, the swap will be remembered and the exam proximity matrix will be updated according to this swap. Due to this, we call this kind of optimization greedy Hill Climbing (HC). Hill Climbing is a neighbourhood search algorithm to locate the best value that can be obtained from a problem space which is around the current solution. It is considered as a local search due to the fact that the algorithm selects better solution which is near to the current obtained solution. The local search will definitely reach a local optimum. However, there is a possibility that there are other local optimums within the search space which is the global optimum, thus requiring a global search. The global optimum will be the main goal of this algorithm. The term 'greedy' here refers to the fact that we always take the best solution whenever it is found in a neighbourhood. A number of repetitions of block

shift and swapping are done in order to ensure the search space is explored in different directions so that the best local optimum or global optimum of the solutions can be found.

Figure 4.4 in Chapter 4 presents an example of a spread matrix. The cost function (2.1) assigns a weight of "16" to exams that are 1 slot apart (entries in the spread matrix (1,2), (2,3), (3,4), etc.) and assigns a weight of "8" to the exams which are 2 slots apart (entries in the spread matrix (1,3), (2,4), (3,5), etc.), and so on. To put it in a slightly more formal way, the weight "16" in the cost function is associated with the "first minor diagonal" entries of the spread matrix; weight "8" is associated with the "second minor diagonal" entries, etc. The potential for the reduction of this cost lies in the possibility of re-ordering the slots so as to replace the big numbers in the first minor diagonal with the smaller entries that are on subsequent minor diagonals.

The reordering of slots has been implemented as a simple greedy optimization process that involved swapping the positions of individual slots and also swapping the positions of groups of adjacent slots. If a swap operation improved the cost function (2.1), the swap was accepted and the exam slots were rearranged accordingly. Recognizing, however, that the greedy optimization may lead to local optima, we have adopted a simple measure of restarting the optimization from several initial orderings of exam slots and picking the best solution from a pre-defined number of runs.

Algorithm 6

```
Generate initial ordering of exam slots
   Repeat for a predefined number of trials
    Shift block of size k in the spread matrix
        Accept block shift if the cost (2.1) is reduced
        Swap individual slots
        Accept the swap if the cost is reduced
        Update the spread matrix with the best schedule
End
```

Figure 3.35: Algorithm for Permutations of Exam Slots Using Greedy Hill Climbing Strategy

It is worth pointing out that while the optimization by permutation of slots does benefit from the prior minimization of the exam-slot conflicts, it does not affect the total count of the exam-slot conflicts because the allocation of individual exams to slots does not change.

A single run of the optimization process outlined in Figure 3.35 on the spread matrix from Figure 4.4 will cause large entries on the first minor diagonal in Figure 4.4 to be replaced with much smaller values that were previously positioned on higher order minor diagonals.

The main core of the algorithm is mainly contributed by three levels of nested loops where each starting point value on the loop is determined by the current value of the outer loop. As the whole process executes, the inner loop gets smaller. The complexity for the algorithm can be easily obtained and would result in $O(n^3)$. This basic algorithm can be further executed within additional loops with a determined number of iterations, with the objective of using the output of the inner loop as a new input for further processing. In our approach we tested various approaches prior to executing the main algorithm, which increases the complexity of the

algorithm to $O(abn^3)$ where the value of a is between 1 and 50 and the value of b is between 1 and 10. Since the values of a and b are small, we can disregard the two coefficients, resulting in the same complexity of $O(n^3)$.

Late Acceptance Hill Climbing

Late Acceptance Strategy was introduced by Burke and Bykov (2008). The strategy concentrates on the timing of a comparison of an accepted solution in an examination scheduling problem. A new solution is compared with a solution found n steps or iterations before, to determine its acceptance as a solution to the problem. In the implementation the researchers created a list of predefined length L as the storage location for all previous solutions. The current solution is only accepted on the list if it meets the minimum or better than the previous solution L steps ago. Whenever a new solution does not meet this criterion, the previous solution that does satisfy it is added to the list to maintain the number of L previous steps. Through their research and participation in the International Optimization Competition in December 2011, they concluded that the Late Acceptance strategy is similar in terms of implementation to greedy Hill Climbing (HC), but much more powerful from the perspective of performance. In Burke and Bykov (2012)'s study, the Late Acceptance Hill Climbing (LAHC) strategy outperformed other one-point search methods; namely, Simulated Annealing, Threshold Accepting and the Great Deluge algorithm.

Based on this finding we have investigated the effectiveness of our approach compared with LAHC, since we are utilizing a greedy Hill Climbing (HC) method as one of our approaches. We hypothesize that if the granulation approach is on a par or better than the Late Acceptance Hill Climbing (LAHC), there will be a very minimal difference in the quality of the solutions generated by the two algorithms, and we could also see which approach would be able to generate better quality examination timetables.

We have implemented the LAHC in our existing Hill Climbing (HC) algorithm implementation by substituting a variable to keep the best cost function with an array of length L. In the current optimization process we are evaluating the moves that we can make in the current solution to find the best move that would result in a better solution; and, once the best move is identified as actual, a new solution is generated. It is in this aspect we implement the LAHC, where a new possible solution from a single slot swap will be evaluated against an accepted solution from a swap L steps earlier.

We differ from Burke and Bykov (2008; 2012) in how we populate or maintain the content of the L list. During each iteration they add to the list with the latest current solution accepted (better solution) and remove the last solution from the list. If the current solution is lower in quality, they add the last current solution accepted to the list and remove the last solution, making the L list change at each iteration and creating the existence of multiple identical values in L.

Our approach, on the other hand, only adds the solution to the *L* if the solution surpasses the existing solution. Burke and Bykov (2008; 2012) implemented the *L* list using a *First In, First Out* approach but we implemented ours as a *Round Robin* list, modifying items at specific locations based on the length of *L* and the number of generations; using the value of the modulus of the number of generations against *L* as the index. We also keep additional information on the list, which is the value of *i* and *j*, indicating the locations of the swap.

The algorithm for Late Acceptance Hill Climbing (LAHC) that we have implemented is as follows:

Algorithm 7

```
For each different starting point (do a block shift of the best solution)
        Set C[0 ...L] to Carter cost for starting point solution
        Set Generation = 0
        For number of repetition
                For k=1 to NumberOfSlot - 1
                        For i = k to NumberOfSlot - 1
                                For j = i+1 to NumberOfSlot
                                        Simulate swap i with j
                                        if NewCost < C[Generation%L]
                                                C[Generation%L]=NewCost
                                                Ci[Generation%L]=i
                                                Cj[Generation%L]=j
                                        End
                                End
                        End
                        If there is an update to C[Generation%L]
                                 Do an actual swap Ci[Generation%L] with
                                 Cj[Generation%L]
                                Generation = Generation +1
                        End
                End
        End
End
```

Figure 3.36: Algorithm for Permutations of Exam Slots Using Late Acceptance Hill Climbing Strategy

The algorithm in general will start with an outer loop that will iterate through the available different starting points. In our approach a starting point is a feasible solution that we have obtained through the allocation method. We have limited the starting points to only 6, which is derived from the best solution found. The second loop is the value that determines the number of trials or cycles to execute the process of the swapping of slots. The main algorithm consists of three levels of loops, one inside another. The two innermost loops function as a permutator that will match every slot in the outer loop, with all other slots starting from i *+1* to the end. The complexity of this loop based on the steps will be *(n-1)+(n-2)+...+0,* which rearranges to the sum of *0* to *n-1; this is T(n)=(n-1)((n-1)+1)/2*. Rearranging this, we can see that *T(n)* will always be smaller than or equal to $1/2(n^2)$, thus giving a complexity of $O(n^2)$. The outermost loop is a shrinking window loop that will reduce the value of *k* each time the loop completes, where *k* will have the value *{n, n-1, n-2,...,1}*. However, with the shrinking window and limitation of each loop the complexity of the algorithm is reduced to $n(n^2) + n\text{-}1(n_2) + n\text{-}2(n^2) \ldots + 0$ which resolves to $O(n^3)$ for the overall algorithm complexity. In reality the number of *n* has a limit on the value with a logical limitation of 365 where it is the number of days in a year. No university will conduct examinations every day in a year, which limits the computational to a maximum of 48,627,125 steps, which is simple for a computer to execute.

*One of the outcomes of the pre-processing stage, the exam spread data structure (namely **spread matrix**), provides the opportunity for re-positioning time slots if the aim is to maximize the gap between consecutive exams. Since the number of available exam slots (as available in the spread matrix) is typically quite low, the optimization of the positions of individual slots can be accomplished by the permutation of rows/columns of the spread matrix and the evaluation of the resulting cost. This second stage of optimization, known as **permutations of exam slots**, is explicitly focused on the minimization of the cost function and brings dividends in terms of having a much smaller slot-optimization problem. The potential for the cost reduction lies in the possibility of re-shuffling the slots to replace the large values in the first minor diagonal with the smaller values on subsequent minor diagonals.*

3.2.4.3 Minimization of Costs via Reassignments of Exams

In the third stage of the optimization, exams that make large contributions to the first minor diagonal entries of the reordered spread matrix are reassigned to slots represented by higher minor diagonals (preferably of order 6 or higher). Shifting an exam from one slot to another has a chain effect. Changes happen not only at the spread matrix level but also in the slot conflict matrix. Alterations of exam slots to reduce the cost function value could further reduce the overall conflict count or increase the value for the current solution. This is because the insertion of an exam to a slot can only happen if the slot exclusively contains exams that do not conflict

with it. This action forces us to reevaluate the slot conflict count, which changes based on the slot location of all exams within the same chain as the shifted exam. The bigger the chain of the exam, the greater the effect it will have on the conflict count. There are two methods of reassignment; single reassignments and group reassignments.

Figure 3.37 below illustrates how a feasible schedule in Figure 3.33 has been further optimized by reassigning exam *E2* to time slot *T5*. At this stage, the gap between exams *E1* and *E2* has been increased greatly (the gap is now 3 slots apart). Note that this is not the final schedule so further optimization will be performed to increase the quality.

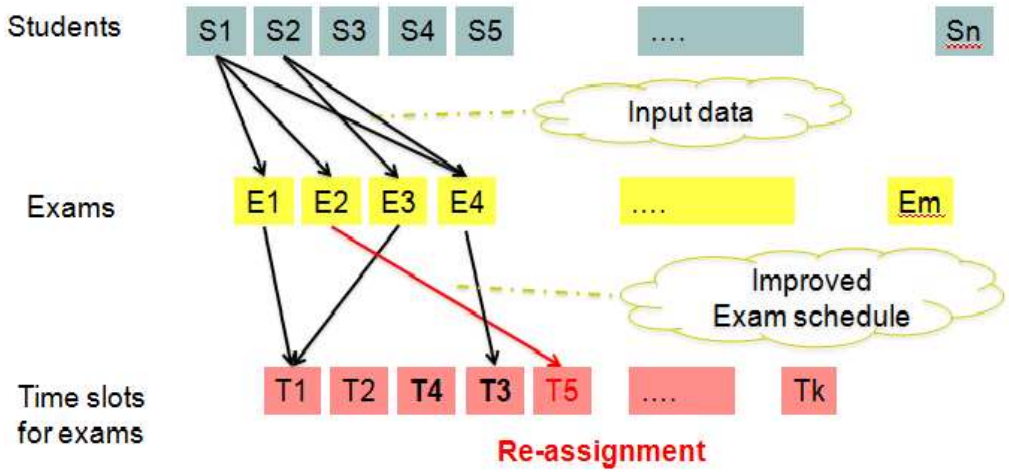


Figure 3.37: An Improved Examination Schedule after Optimization (Reassignment of Exam)

The single reassignments optimization move throughout the search space to identify an exam that has the biggest reduction to the cost function (2.1) if it were to be moved to other slots. The algorithm looks for a conflict-free slot which leads to the biggest cost reduction for all exams.

The process of identifying the possible slots and calculating the cost function contribution is made simple by a data structure that combines the slot location and the penalty values generated by each exam for a slot.

The group reassignments optimization moves throughout the search space to identify an exam which could lead to the biggest reduction to the cost function. The optimization process evaluates the reduction from moving an exam to all other slots, and the best combination or total reduction configuration will be selected as the move that will be executed.

During the process of optimization the generated possible moves are evaluated against a history of moves two steps behind. The purpose of this is to eliminate possible cyclic moves in the optimization process. The group reassignments move exams to another slot if a reduction can be obtained. This may push other exam(s) out of the selected slot to alternative slots. There is a possibility that these exams switch slots and keep on giving an improvement to the Carter cost (2.1); thus, keeping the optimization process ongoing. This will not stop if these two exams keep exchanging slots. This kind of move must be identified to eliminate infinite swapping. The process continues until there is no more improvement available and the number of iterations is more than half of the number of exams.

Both the single and group reassignments start by evaluating each exam, one at a time, and looking for possible slot locations that could accept the exam without any clashes. The main difference is in the evaluation criteria to shift and the number of exams for every shift. The

single reassignment will end up with solutions within the local optima due to the minor changes made to the initial placement of exams.

However, the group optimization has the possibility of moving solutions out from the local optima, concluding in a better result. This is due to the larger changes made at each step where all exams are evaluated and shifted at each cycle.

The effect of reassignments on the schedule is that the cost (2.1) will be reduced at the expense of some increase in the total number of slot conflicts. The pseudocode for the reassignment process is outlined in Figure 3.38.

Algorithm 8

```
Obtain the number of slots from the spread matrix
Obtain the number of exams from the exam conflict matrix
Read exam-to-slot allocation
Read Slot Conflict Matrix
For all exams
        For all slots
           Find the most beneficial exam to reassign by calculating the potential cost
           (2.1)
           If improvement is possible
                 Reassign the exam to the new slot
                 Update Allocation Flag, Slot Conflict Matrix
            End
        End
End
```

Figure 3.38: Algorithm for Reassigning Exams

The complexity for the algorithm can be easily obtained, which would result in $O(mn)$, where m is the number of exams and n is the number of slots. This can be further simplified to $O(n^2)$, with a limiting value of either the number of exams or the number of slots.

In the third optimization stage, the information of the reordered spread matrix is utilized to further reduce the cost of the schedule. Exams that make a large contribution to the first minor diagonal entries are reassigned to slots represented by higher minor diagonals (preferably of order 6 or higher). This procedure is expected to reduce more of the existing cost.

3.3 Mathematical Formulation Based on the Proposed Approach

In this section, the examination scheduling problem is represented using mathematical formulation according to our proposed approach. Recall that we generated our solution by assigning each exam in the problem to a time slot by checking certain criteria (e.g. the value in the Exam Conflict Matrix) and ensuring certain conditions were fulfilled. The complete formulations are shown below:

E *is a set of Exams and* $\neg \{\}$

N_E *is the number of Exams or* $|E|$

E_i *is an exam where* $i > 0$, $I \leq N_E$ *and* $E_i \epsilon E$

S *is a set of Students and* $\neg\{\}$

N_s *is the number of Students or* $|S|$

S_i *is a student where* $i > 0$, $i \leq N_s$ *and* $S_i \epsilon S$

T *is a set of Slot and* $\neg \{\}$

N_T *is the number of Slot or* $|T|$

T_i *is a slot where* $i > 0$, $i \leq N_T$ *and* $T_i \epsilon T$

L_i is a set specifying the Enrolment of Student S_i , where the element in $L_i \in E$

$ECM = (ECM_{mn})_{|E| \times |E|}$ is the Exam Conflict Matrix where each element denoted by ECM_{mn} is the negative summation of S_h that fulfil $E_m \in L_h$, $E_n \in L_h$, $E_m \in E$ and $E_n \in E$

A_m is the allocation of an exam to a slot p. $A_m = p$, $T_p \in T$.

Therefore, an Examination Scheduling Problem is the problem of allocating E to a T that fulfils:

$\forall E_i \in E$, $\exists T_j$ where $ECM_{ik} = 0$, $\forall E_k$, where $k=\{1,\ldots,|E|\}-\{i\}$ and $A_j = A_k$.

3.4 Recap of the Proposed Approach

As a summary, the proposed method proceeds in the following stages:

1) Problem domain transformation from student-exam to exam conflict and spread matrix data space
2) Generation of a feasible schedule
3) Minimization of the overall slot conflicts
4) Minimization of the schedule cost by slot swapping
5) Minimization of the schedule cost by exam reallocation
6) Repetition of stages 4 and 5 until there is no improvement in the schedule cost.

CHAPTER 4

Experiments, Results, and Discussions

This chapter discusses the results and findings of experiments that were performed on benchmark datasets using the proposed approach. The chapter starts by presenting the outcome of each phase involved in the proposed framework one by one, before the complete set of results for all datasets is presented. Afterwards, the results obtained by the proposed approach are compared with other constructive methods that have been reported in the literature.

4.1 Experiments and Results for Benchmark Datasets

This chapter will discuss the experiments that were performed and the results that were obtained based on the proposed approach described in the previous chapter and applied to benchmark datasets. This includes the pre-processing of data, scheduling, and the optimization stage.

4.1.1 Pre-processed Data

The pre-processed data that was generated in the early stages include: the exam conflict matrix, conflict chains, and the spread matrix. All of these pre-processed data were utilized to a high degree in the scheduling and optimization process at the later stages. This chapter provides also illustrations and explanations of these data structures.

Exam Conflict Matrix

An exam conflict matrix produced by the pre-processing stage, as discussed in the previous chapter, is illustrated in Figure 4.1. As can be seen, it is a square matrix containing –s value at position i,j where s is the number of students causing conflict between exam i and j. The matrix is symmetrical with diagonal elements (shaded in grey) by definition equal to 0 because an exam cannot be in conflict with itself. The last row and column indicate the exam number.

0	-19	-23	-291	-35	-247	-16	-12	-3	-18	-275	-20	-303	1
-19	0	-174	-232	-171	-192	-29	-6	-11	-132	-243	-170	-252	2
-23	-174	0	-2	-128	-6	-33	0	-12	-133	-28	-168	-19	3
-291	-232	-2	0	-75	-435	-19	-12	-2	-3	-506	-11	-535	4
-35	-171	-128	-75	0	0	0	0	0	-127	-71	-129	-73	5
-247	-192	-6	-435	0	0	0	0	0	-5	-426	-6	-439	6
-16	-29	-33	-19	0	0	0	0	0	-2	-21	-42	-19	7
-12	-6	0	-12	0	0	0	0	-1	-24	-11	-6	-29	8
-3	-11	-12	-2	0	0	0	-1	0	-1	-3	-15	-1	9
-18	-132	-133	-3	-127	-5	-2	-24	-1	0	-6	-127	-27	10
-275	-243	-28	-506	-71	-426	-21	-11	-3	-6	0	-26	-516	11
-20	-170	-168	-11	-129	-6	-42	-6	-15	-127	-26	0	-3	12
-303	-252	-19	-535	-73	-439	-19	-29	-1	-27	-516	-3	0	13
1	2	3	4	5	6	7	8	9	10	11	12	13	0

Figure 4.1: An Example of an Exam Conflict Matrix

In this particular example, the matrix has been generated by a dataset containing 13 exams. Each value in the cell -s, which is at the intersection of row i and column j (or vice versa) means that there are s students taking both exam i and j. As an example, the value -19 at the intersection of column 1 and row 2 (or vice versa) means there are 19 students causing conflict between exam i and j. On the other hand, the fact that the value zero is shown at the intersection of column 3 and row 8 demonstrates that exam 3 and exam 8 do not clash.

Useful information is provided by this matrix during the scheduling process; in particular the zero value found during traversing of the matrix, immediately indicates that both exams can be scheduled concurrently in the same timeslot.

Conflict Chains

A representative section of the output of the conflict chain generation algorithm (Rahim *et al.,* 2009), described in Section 3.2.2.2, is shown in Figure 4.2. The chain label is given in the first row and the exam number is indicated in the first column. It can be seen that conflict chains 1, 3, and 7 share some of the exams between them. For example, exam 309 is shared between conflict chain 1 and 3; exam 311 is shared between conflict chain 1 and 5; and exam 440 is shared between conflict chains 1 and 7. This means that conflict chains 1, 3, and 7 can be justifiably merged into one conflict chain. It is worth noting that the merged conflict chain of 1 and 7

will also include exams 317 and 321 even if they were previously only displayed in conflict chain 7.

The resulting merged conflict chain is illustrated in Figure 4.3 under the chain label 1. Conflict chains 2, 4, and 6 are unaffected by this post processing as they do not have any exams in common with those from chain 1. In particular, it should be noted that exam 323 is assigned to time slot 8 in conflict chain 6 but, since it is not in conflict with any of the exams in chain 1, it has the label 0 in the merged conflict chain 1.

Furthermore, the length of the conflict chain, measured as the number of necessary time slots to schedule the exams in the chain is an immediate indication of the difficulty of the specific scheduling task.

	1	2	3	4	5	6	7
309	11	0	11	0	0	0	0
310	2	0	0	0	0	0	0
311	7	0	0	0	7	0	0
312	6	0	0	0	0	0	0
313	1	0	0	0	0	0	0
314	1	0	1	0	0	0	0
315	3	0	0	0	0	0	0
316	19	0	19	0	0	0	0
317	0	0	0	0	0	0	1
318	6	0	0	0	0	0	0
319	11	0	0	0	0	0	0
320	10	0	10	0	0	0	0
321	0	0	0	0	0	0	2
322	11	0	0	0	0	0	0
323	0	0	0	0	0	8	0
324	10	0	10	0	0	0	0
325	11	0	11	0	0	0	0
326	2	0	0	0	0	0	0
327	9	0	0	0	0	0	0
	:	:	:	:	:	:	:
439	4	0	4	0	0	0	0
440	1	0	0	0	0	0	1
441	13	0	13	0	0	0	0

Figure 4.2: Conflict Chains before Merging

	1	2	3	4	5	6	7
309	11	0	0	0	0	0	0
310	2	0	0	0	0	0	0
311	7	0	0	0	0	0	0
312	6	0	0	0	0	0	0
313	1	0	0	0	0	0	0
314	1	0	0	0	0	0	0
315	3	0	0	0	0	0	0
316	19	0	0	0	0	0	0
317	1	0	0	0	0	0	0
318	6	0	0	0	0	0	0
319	11	0	0	0	0	0	0
320	10	0	0	0	0	0	0
321	2	0	0	0	0	0	0
322	11	0	0	0	0	0	0
323	0	0	0	0	0	8	0
324	10	0	0	0	0	0	0
325	11	0	0	0	0	0	0
326	2	0	0	0	0	0	0
327	9	0	0	0	0	0	0
:	:	:	:	:	:	:	:
439	4	0	0	0	0	0	0
440	1	0	0	0	0	0	0
441	13	0	0	0	0	0	0

Figure 4.3: Conflict Chains after Merging

By inspecting the conflict chains, three immediate observations can be made. If the number of available time slots is smaller than the length of the largest conflict chain, the scheduling problem is infeasible. If the number of available time slots is not much greater than the length of the longest conflict chain then the problem is heavily constrained and the quality of the resulting timetable, measured by the cost function, might be expected to be low. On the other hand, if the number of the available time slots is significantly greater than the length of the longest conflict chain then high quality (low cost) solutions can be expected.

Spread Matrix

An example of a spread matrix (Rahim *et al.,* 2009; Rahim *et al.,* 2012) generated by the algorithm presented in the previous chapter is shown in Figure 4.4. As previously mentioned, a spread matrix is a square matrix of dimension s, where s is the number of slots. Entries in the spread matrix at position (p,q) represent the number of students who took an exam from both slot p and slot q. The last row and column in the matrix indicate the slot number. The matrix is symmetrical with diagonal elements omitted because only one exam can be taken by students in any given exam slot. The spread matrix presented below has been generated for a dataset which requires 10 slots to schedule all exams.

0	1044	1108	918	948	708	628	47	222	7	1
1044	0	1349	1119	1302	593	753	118	322	9	2
1108	1349	0	1282	1198	575	786	166	342	9	3
918	1119	1282	0	921	518	656	95	181	33	4
948	1302	1198	921	0	684	733	159	194	33	5
708	593	575	518	684	0	546	92	23	45	6
628	753	786	656	733	546	0	79	140	43	7
47	118	166	95	159	92	79	0	35	12	8
222	322	342	181	194	23	140	35	0	25	9
7	9	9	33	33	45	43	12	25	0	10
1	2	3	4	5	6	7	8	9	10	0

Figure 4.4: A Spread Matrix for a Dataset with 10 Time Slots

It can be observed that there are 1044 students taking exams in time slot 1 and 2 and 1,108 students taking exams in time slot 1 and 3, etc. The cost function (2.1) assigns a weight of “16” to exams that are 1 slot apart (i.e. blue cells in the spread matrix (1,2), (2,3), (3,4) etc.) and a weight of “8” is assigned to an exam 2 slots apart (i.e. green cells in the

spread matrix (1,3), (2,4), (3,5), etc.), and so on (with yellow, pink, purple, and grey assigned the weights 4, 2, 1, and 0 respectively).

Useful information will be provided by this matrix during optimization of the schedules at a later stage. This is made possible by using the background knowledge of the structure of the cost function (2.1), the renumbering of the time slots could be done to maximize the spread of examinations.

4.1.2 Schedules Generated

4.1.2.1 Initial Feasible Schedule

The initial feasible schedule generated at this stage is based on the allocation method discussed in Chapter 3. The output is an allocation flag, exam-to-slot vector in which the slot number for all exams is contained. At this point, the number of slots could be determined by the maximum value in the allocation flag. A representative section of an allocation flag (*allocflag*) for a dataset with 181 exams (yorf83), before and after respectively performing the backtracking process, is shown in Figure 4.5 and Figure 4.6.

The slot number for each exam is represented by the numbers in the column of both Figures 5.5 and 5.6 (numbered outside the column for the purpose of reference). The number of slots required to schedule all the exams in both allocation flags (*allocflag)* is 22 and 21 respectively.

1	11
2	13
3	7
4	16
5	11
6	1
:	:
:	:
54	14
55	1
56	13
57	12
58	10
59	5
:	:
:	:
71	5
72	10
73	7
74	4
75	13
:	:
:	:
98	12
99	14
100	3
101	20
102	17
103	5
104	19
:	:
:	:
177	21
178	21
179	22
180	7
181	19

Figure 4.5: *allocflag* for yorf83 before backtracking

1	11
2	13
3	7
4	16
5	11
6	1
:	:
:	:
54	14
55	1
56	14
57	12
58	10
59	5
:	:
:	:
71	5
72	10
73	7
74	13
75	13
:	:
:	:
98	12
99	14
100	5
101	17
102	3
103	20
104	19
:	:
:	:
177	21
178	21
179	4
180	7
181	19

Figure 4.6: *allocflag* for yorf83 after backtracking

Backtracking has been deployed after scheduling in order to reduce the number of slots, as described in Section 3.2.3.1.2. Figures 4.5 and 4.6 illustrate the effect of backtracking by showing that after reassignment of exams to time slots only 21 slots are required to schedule all exams for yor-f-83 dataset (instead of 22 slots required before doing backtracking). Backtracking plays an important role to satisfy the number of slots stipulated by the problem statement.

The number of slots generated together with the cost obtained on the initial feasible schedules before and after performing backtracking (which tries to eliminate the last slot) on the University of Nottingham and University of Toronto datasets are given in Table 4-1.

Based on this table, five datasets managed to reduce the number of slots (bold numbers indicate the reduction). The proposed backtracking tries to eliminate the last slot in the initial feasible schedule generated by allocating the existing exams in the last slot to other slots while maintaining the feasibility, hence only a reduction of one slot can be seen in the five successful cases (Nott, Car-s-91, Pur-s-93, Tre-s-92 and Yor-f-83). It is worth highlighting here that the number of slots in our initial feasible schedules is already quite small (the same or smallest than the required number of slots for all but the yor-f-83 problem), therefore a significant reduction in terms of the number of slots is not critical. However, any reduction represents an advantage because it allows extra buffering space during permutations of exam slots at the later optimization stage.

Table 4-1: Number of Slots for Nott and Toronto Datasets Before and After Performing Backtracking

Name of Data set	Slots Required (as in the literature)	Before Performing Backtracking		After Performing Backtracking	
		Number of Slots Based on *allocflag*	Initial Cost Before Optimization	Number of Slots Based on *allocflag*	Initial Cost Before Optimization
Nott	23	19	38.99	**18**	38.33
Car-s-91	35	33	11.77	**32**	11.79
Car-f-92	32	31	9.43	31	9.43
Ear-f-83	24	24	72.69	24	72.69
Hec-s-92	18	18	22.83	18	22.83
Kfu-s-93	20	19	37.79	19	37.79
Lse-f-91	18	18	23.77	18	23.77
Pur-s-93	42	37	14.91	**36**	14.87
Rye-f-92	23	22	31.50	22	31.50
Sta-f-83	13	13	201.95	13	201.95
Tre-s-92	23	23	14.81	**22**	14.12
Uta-s-92	35	34	8.71	34	8.71
Ute-s-92	10	10	60.71	10	60.71
Yor-f-83	21	22	59.04	**21**	57.19

The backtracking procedure tested on Toronto benchmark datasets has successfully reduced the number of slots for some datasets. In the yor-f-83 dataset, this is prioritized in order to satisfy the minimum number of slot restrictions imposed in the problem. For all cases that recorded a reduction in slots, the cost after backtracking was further reduced, which is an added advantage to preparing a schedule with extra buffering space for slot permutations in succeeding optimizations. At this stage the exam schedule generated is always feasible but not necessarily optimal.

4.1.3 Improved Quality Schedules via Optimization

The optimization documented in this section shows improvement of the initial feasible schedule that was generated in the previous section. It includes the minimization of the overall slot conflicts, minimization of the schedule cost by slot swapping, and minimization of the schedule cost by exam reallocation (Rahim *et al.,* 2012). The results obtained by all these processes are given in sections 4.1.3.1, 4.1.3.2 and 4.1.3.3 respectively.

4.1.3.1 Minimization of Total Slot Conflicts

The first step in the optimization stage is to minimize the total slot conflicts as described in Section 3.2.4.1. Table 4-2 shows that the technique of minimizing the total slot conflicts as well as the cost of the exam schedule has been shown to be effective. This stage can be considered as an enhancement of the potential for subsequent minimization of the cost of the schedule.

Table 4-2: Results after Performing the Minimization of Total Slot Conflicts Procedure on Nott and Toronto Datasets

Dataset	**Re-quired No of Slots**	**Initial Cost**	**Total Slot Conflicts before**	**Total Slot Conflicts after**	**Cost after reduction of slot conflicts**
Nott	23	38.99	8589	8090	31.95
car-s-91 (I)	35	11.77	17169	16665	10.43
car-f-92 (I)	32	9.43	12332	12217	8.89
ear-f-83(I)	24	72.69	3582	3544	62.57
hec-s-92(I)	18	22.83	1263	1243	25.15
kfu-s-93	20	37.79	4616	4544	29.89
lse-f-91	18	23.77	3739	3685	21.35
pur-s-93 (I)	42	14.91	49821	49470	14.07
rye-f-92	23	31.50	7178	6782	26.05
sta-f-83(I)	13	201.95	1507	1505	193.47
tre-s-92	23	14.81	4392	4251	13.25
uta-s-92(I)	35	8.71	15859	15416	8.28
ute-s-92	10	60.71	1200	1149	46.57
yor-f-83 (I)	21	59.04	3336	3256	56.31

Table 4-2 documents that the minimization of total slot conflicts reduces simultaneously the total number of slot conflicts and the cost of the schedule. By reducing the total slot conflicts a greater packing of conflicting exams was achieved and by implication, an increased possibility of separating the slots that have the largest number of conflicting exams was also obtained. This first optimization stage can be considered as an enhancement of the potential for further reduction of the cost of the solution.

4.1.3.2 Cost Reduction via Permutation of exam slots

This subsection documents results obtained by the permutation of exam slots obtained by the allocation method that generated the initial feasible schedule..

Costs Produced By Method 1 versus Method 2

Methods 1 and 2, described in subsection 3.2.4.2.1 and 3.2.4.2.2 respectively (Rahim *et al.,* 2009), have been evaluated on the University of Nottingham dataset and the results are presented and discussed below. Figure 4.7 is the representative section of the first 6 slots of the spread matrix for Nott dataset. The total number of slots for this dataset is 18 and 23 for uncapacitated and capacitated problems respectively.

0	1454	1360	1717	1276	1006	1
1454	0	1355	1634	1085	997	2
1360	1355	0	1392	1158	947	3
1717	1634	1392	0	1529	1446	4
1276	1085	1158	1529	0	1120	5
1006	997	947	1446	1120	0	6
1	2	3	4	5	6	0

Figure 4.7: Initial Ordering of the Spread Matrix for the First 6 Slots for the Nottingham Dataset

The first six rows and columns of the re-numbered spread matrix using Method 1 and 2 are shown in Figure 4.8 and Figure 4.9 respectively. However, some of the rows and columns represented in Figures 4.8 and 4.9 do not appear in Figure 4.7 because their

corresponding time slot number is greater than 6. Nevertheless the sample spread matrices serve to illustrate the main characteristics of the two methods.

0	1006	1360	1276	1454	1717	1
1006	0	947	1120	997	1446	2
1360	947	0	1158	1355	1392	3
1276	1120	1158	0	1085	1529	4
1454	997	1355	1085	0	1634	5
1717	1446	1392	1529	1634	0	6
1	2	3	4	5	6	0

Figure 4.8: The New Arrangements of the Initial Ordering of the Spread Matrix after Applying Method 1

0	1006	1717	1360	1454	1276	1
1006	0	1446	947	997	1120	2
1717	1446	0	1392	1634	1529	3
1360	947	1392	0	1355	1158	4
1454	997	1634	1355	0	1085	5
1276	1120	1529	1158	1085	0	6
1	2	3	4	5	6	0

Figure 4.9: The New Arrangements of the Initial Ordering of the Spread Matrix after Applying Method 2

The cost function (2.1) evaluated with the optimization of the exam spread using Method 1 and Method 2 is presented in Table 4-3.

Table 4-3: Cost Functions Before and After Considering the Spread Information for the Uncapacitated Nott Dataset.

No of Slots	18
Cost Function with Original Ordering of Time Slots	43.91
Cost Function After Rearrangement of Slots Using Method 1	29.03
Improvement Percentage (%)	33.89
Cost Function After Rearrangement of Slots Using Method 2	**24.18**
Improvement Percentage (%)	44.93

Furthermore, the optimization of the spread matrix by re-numbering of exam slots leads to a significant improvement of the cost function. We considered the smallest number of time slots that allows generation of a feasible schedule. As such, the cost function is large because there is little room for manoeuver as far as moving time slots around is concerned.

An alternative version of the Nottingham exam-scheduling problem involves, on the one hand, a relaxation of the constraint on the number of time slots from 18 to 23 and the introduction of an additional constraint on the number of students taking exams in any of the time slots (maximum number 1550). When the same cost function (2.1) is used, and the results for the original ordering of time slots are evaluated, the results of the optimized ordering obtained by using Method 1 and 2 are presented in Table 4-4.

Table 4-4: Cost Functions Before and After Considering the Spread Information for the Capacitated Nott Dataset.

No of Slots	23
Cost Function with Original Ordering of Time Slots	22.51
Cost Function After Rearrangement of Slots Using Method 1	21.29
Improvement Percentage (%)	5.42
Cost Function After Rearrangement of Slots Using Method 2	**19.61**
Improvement Percentage (%)	12.88

The inspection of the spread matrix that was generated by both methods has revealed that the first method tends to over-emphasize the selection of small spread values on the first minor diagonal and, by the time the few remaining time slots are dealt with by the optimization process, it is forced to leave the high spread values at the bottom right section of the first minor diagonal by the capacity constraints. In contrast Method 2 takes a more balanced approach to optimizing the spread values and is less affected by the capacity constraint, thus producing a lower overall cost.

Costs Produced By Greedy Hill Climbing

The next method in which the concept of permutations of exam slots is utilized is the Greedy Hill Climbing algorithm (Rahim *et al.,* 2009; Rahim *et al.,* 2012), as described in section 3.2.4.2.3. Illustrated in Figure 4.10 below is an example of a spread matrix which requires 10 slots to schedule all the exams in the initial feasible solution.

0	1044	1108	918	948	708	628	47	222	7	1
1044	0	1349	1119	1302	593	753	118	322	9	2
1108	1349	0	1282	1198	575	786	166	342	9	3
918	1119	1282	0	921	518	656	95	181	33	4
948	1302	1198	921	0	684	733	159	194	33	5
708	593	575	518	684	0	546	92	23	45	6
628	753	786	656	733	546	0	79	140	43	7
47	118	166	95	159	92	79	0	35	12	8
222	322	342	181	194	23	140	35	0	25	9
7	9	9	33	33	45	43	12	25	0	10
1	2	3	4	5	6	7	8	9	10	0

Figure 4.10: An Example of a Spread Matrix with 10 Slots Before Performing Greedy Hill Climbing Procedure

Assuming that the total number of students is 2,749 the cost function (2.1) is evaluated as:

[[(1044 + 1349 + 1282 + 921 + 684 + 546 + 79 + 35 + 25) * 16] +

[(1108 + 1119 + 1198 + 518 + 733 + 92 + 140 + 12)* 8] +

[(918 + 1302 + 575 + 656 + 159 + 23 + 43) * 4] +

[(948 + 593 + 786 + 95 + 194 + 45) * 2] +

[(708 + 753 + 166 + 181 + 33) * 1]] / 2749

= 56.99

A single run of the Greedy Hill Climbing algorithm on the spread matrix from Figure 4.10 has resulted in the spread matrix that is presented in Figure 4.11.

0	575	342	1198	9	1282	166	1108	786	1349	3
575	0	23	684	45	518	92	708	546	593	6
342	23	0	194	25	181	35	222	140	322	9
1198	684	194	0	33	921	159	948	733	1302	5
9	45	25	33	0	33	12	7	43	9	10
1282	518	181	921	33	0	95	918	656	1119	4
166	92	35	159	12	95	0	47	79	118	8
1108	708	222	948	7	918	47	0	628	1044	1
786	546	140	733	43	656	79	628	0	753	7
1349	593	322	1302	9	1119	118	1044	753	0	2
3	6	9	5	10	4	8	1	7	2	0

Figure 4.11: An Example of a Spread Matrix with 10 Slots after Performing the Greedy Hill Climbing Procedure

The large entries on the first minor diagonal in Figure 4.10 are replaced with much smaller values that were previously positioned on higher order minor diagonals. The cost (2.1) after the permutations of slots is:

$$[[(575 + 23 + 194 + 33 + 33 + 95 + 47 + 628 + 753) * 16] +$$
$$[(342 + 684 + 25 + 921 + 12 + 918 + 79 + 1044)* 8] +$$
$$[(1198 + 45 + 181 + 159 + 7 + 656 + 118) * 4] +$$
$$[(9 + 518 + 35 + 948 + 43 + 1119) * 2] +$$
$$[(1282 + 92 + 222 + 733 + 9) * 1]] / 2749$$
$$= 31.81$$

If the permutations of exams slots based on the Greedy optimization can lead to local optima then the sensitivity of this optimization to the number of starting points is investigated, so as to ensure sufficient exploration of the search space and promote the

convergence to the global optimum. However, no claim is made regarding the exhaustive exploration of the search space and instead the plots of the convergence trajectories in the "exam conflict – schedule cost" space are offered as an indication of the robust performance of the proposed method.

Different Parameters for Permutations of Slots

Different combinations of parameters have been tested in order to find the ideal or sufficient combinations that would lead to local optima. The numbers 6, 9, and 12 have been used as starting points; and experiments for iterations 4, 8, 10, and 12 have been performed. All combinations were tested and the results for all datasets are recorded in Table 4-5.

Table 4-5: Optimized number of starting points and repetitions of the permutations of exam slots for different benchmark problems.

Dataset	Carter Cost Before Permutations of Slots (Before Optimizations)	Number of Starting Points Providing Best (local) Optimum	Number Of Repetitions Providing Best (local) Optimum	Carter Cost After Permutations of Slots	CPU Time (seconds)
Nott	31.95	6	6	10.74	15.95
car-s-91	10.43	9	6	6.36	201.50
car-f-92	8.89	12	4	5.29	101.72
ear-f-83	62.57	6	4	39.54	18.59
hec-s-92	25.15	6	4	11.49	10.77
kfu-s-93	29.89	12	4	15.91	18.25
lse-f-91	21.35	6	4	14.11	9.14
pur-s-93	14.07	9	6	6.64	277.27
rye-f-92	26.05	6	4	12.34	18.70

sta-f-83	193.47	9	6	173.36	6.05
tre-s-92	13.25	6	4	9.75	14.42
uta-s-92	8.28	9	4	4.28	149.08
ute-s-92	46.57	6	4	30.85	1.34
yor-f-83	56.31	6	4	39.94	34.45

The study indicated that a combination of 12 starting points and 6 iterations provided the best (sub-optimal) results on the benchmark dataset and that the increase of the number of iterations did not produce any improvement in cost. In order to further enhance the exploration of the search space, 24 random starting points and 6 iterations were adopted in all subsequent experiments. This was made possible because the optimization of slot ordering is relatively inexpensive in terms of computational power.

The results for all datasets utilizing 24 starting points and 6 iterations are therefore presented in Table 4-6. It should be noted that the total number of slot conflicts is maintained after the permutations of exam slots because the allocation of individual exams to slots was not changed.

Table 4-6: Results Before and After Performing Permutation of Exam Slots on Nott and Toronto Datasets

Dataset	Initial Cost	Cost Before Per-mutations of Exam Slots	Total Slot Conflicts	Cost After Per-mutations of Exam Slots	Total Slot Conflicts
nott	31.95	31.95	8090	10.94	8090
car-s-91 (I)	10.43	10.43	16665	6.26	16665
car-f-92 (I)	8.89	8.89	12217	5.36	12217
ear-f-83(I)	62.57	62.57	3544	40.45	3544
hec-s-92(I)	25.15	22.55	1263	12.52	1263
kfu-s-93	29.89	29.89	4544	16.06	4544
lse-f-91	21.35	22.42	3739	14.63	3739
pur-s-93 (I)	14.07	14.27	49821	6.69	49821
rye-f-92	26.05	28.55	7178	12.68	7178
sta-f-83(I)	193.47	193.47	1505	158.43	1505
tre-s-92	13.25	13.25	4251	9.84	4251
uta-s-92(I)	8.28	8.28	15416	4.24	15416
ute-s-92	46.57	46.57	1149	29.82	1149
yor-f-83 (I)	56.31	56.31	3256	43.36	3256

4.1.3.2.1 Costs Produced By Late Acceptance Hill Climbing (LAHC)

In this section we document the results obtained by implementing the LAHC strategy described in section 3.2.4.2.4. Different L was used in order to examine the effectiveness of increasing the value. According to Burke and Bykov (2008), the increase of L would increase the computational cost and simultaneously help to achieve better solutions.

As such, a lower Carter cost (2.1) is expected to be obtained with an increased *L* value.

Table 4-7: Results before and after Performing LAHC Permutations of Exam Slots on Nott and Toronto Datasets

Dataset	Costs Produced By Permutations of Slots in the First Stage Optimization *(not the final costs)*				
	Traditional Greedy Hill Climbing	Late Acceptance Hill Climbing *(with different length (L))*			
		L = 1	*L = 5*	*L = 10*	*L = 50*
nott	10.6031	10.6031	10.6031	10.6031	***10.5968***
car-s-91 (I)	6.2564	6.2564	6.2564	6.2564	***6.2304***
car-f-92 (I)	***5.3625***	***5.3625***	***5.3625***	***5.3625***	5.3745
ear-f-83(I)	40.4516	40.4516	39.96	***39.7813***	40.5698
hec-s-92(I)	12.519	12.519	***12.3234***	12.481	12.4676
kfu-s-93	16.0615	16.0615	***15.7846***	16.0578	***15.7846***
lse-f-91	14.6321	14.6321	***14.5238***	14.5873	14.5873
pur-s-93 (I)	***6.3294***	***6.3294***	6.6496	6.6496	6.5603
rye-f-92	12.6768	12.6768	***12.481***	***12.481***	***12.481***
sta-f-83(I)	***158.4157***	***158.4157***	***158.4157***	158.639	158.639
tre-s-92	***9.8375***	***9.8375***	***9.8375***	***9.8375***	***9.8375***
uta-s-92(I)	4.2357	4.2357	4.2133	***4.1836***	4.2037
ute-s-92	***29.2862***	***29.2862***	***29.2862***	***29.2862***	***29.2862***
yor-f-83 (I)	***43.3549***	***43.3549***	***43.3549***	***43.3549***	43.5218

The results obtained by LAHC slot permutations have been recorded and compared with the results obtained by traditional Greedy Hill Climbing as shown in the above table. By analyzing the results, it is clearly shown that the *L* value does not guarantee a better result when increased. In two datasets, nott and car-s-91(I), LAHC managed to reduce

the Carter cost (2.1) when L was increased to 50. In these examples, the increase in L reduced the initial cost which was stagnant when using L = *1, 5,* and *10*. However, in many more datasets for example car-f-92 (I), ear-f-83(I), hec-s-92(I), lse-f-91, pur-s-93 (I), sta-f-83(I), uta-s-92(I), and yor-f-83 (I) the cost increased when L increased. For the two other datasets tre-s-92 and ute-s-92 the cost was the same for different L values that were tested. This probably happened due to the configurations of examinations allocations (slots ordering) which reached the local or global optimum contributed by the small search space.

In comparison to the Greedy Hill Climbing (GHC), we can see in some datasets, LAHC outperformed HC, but in some cases they are equal. Based on this finding, it was quite difficult to predict the quality of solutions using the LAHC, therefore the greedy HC was used in the optimization stage of this study.

The second stage of optimization that is proposed is the ***minimization of costs via the permutations of exam slots.*** *This procedure was performed to re-order exam slots in the spread matrix with the aim of minimizing the large elements in the first minor diagonal by replacing them with smaller entries from subsequent minor diagonals. A few different methods have been implemented, but it was decided that Greedy Hill Climbing was the best procedure for obtaining effective and consistent solutions. From the results presented, it can be clearly seen that the cost was greatly reduced after this procedure was applied to the schedule. The approach of repeating and restarting the search from different starting points was worthwhile in obtaining optimized schedules.*

4.1.3.3 Cost Reduction via Reassignments of Exams

After the permutations of slots was performed on the feasible schedule, the next stage of optimization was the reassignment of exams between slots, as described in section 3.2.4.3. The best results obtained by this process (i.e. either single or group reassignments) are recorded in Table 4-8. It should be noted that the total slot conflicts for all datasets increased after the reassignment process. This is due to alterations to the allocation of exams to slots during this process.

The results presented later in this chapter will show in detail the cost obtained for each type of reassignment in the benchmark datasets.

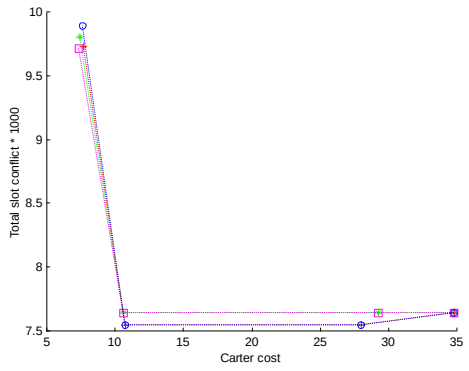
Table 4-8: Results before and after Performing Reassignments of Exams Between Slots

Dataset	Initial Cost	Cost Before Reassignment	Total Slot Conflicts	Cost After Reassignment	Total Slot Conflict
nott	38.99	10.94	8090	7.34	9979
car-s-91 (I)	11.77	6.26	16665	5.19	18847
car-f-92 (I)	9.43	5.36	12217	4.52	13558
ear-f-83(I)	72.69	40.45	3544	37.57	3707
hec-s-92(I)	22.83	12.52	1263	11.85	1266
kfu-s-93	37.79	16.06	4544	14.36	5174
lse-f-91	23.77	14.63	3739	12.41	4077
pur-s-93 (I)	14.91	6.69	49821	4.92	60005
rye-f-92	31.50	12.68	7178	9.80	7664
sta-f-83(I)	201.95	158.43	1505	158.25	1507
tre-s-92	14.81	9.84	4251	8.77	4714
uta-s-92(I)	8.71	4.24	15416	3.59	16792
ute-s-92	60.71	29.82	1149	27.37	1274
yor-f-83 (I)	59.04	43.36	3256	41.35	3412

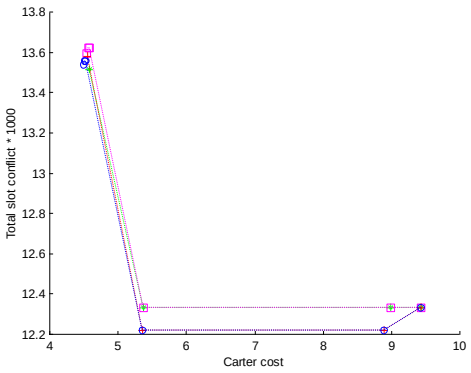
The third stage of optimization: the reassignment of exams should also be considered worth executing because from the results that were presented it can be clearly shown the cost was further reduced for all datasets after its execution. This has proven that the reassignment of exams that make a large contribution to the first minor diagonal entries in the spread matrix should be considered a reliable process. It indirectly demonstrates that pre-processing has supplied valuable information, i.e. regarding the spread matrix. Furthermore, the optimization was assisted by its intelligent exploitation of the available information.

4.1.4 Summary of Results and Graphs Produced For Benchmark Datasets Using Proposed Approach

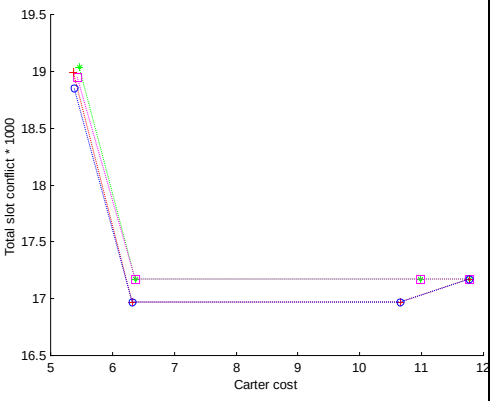
The results for all the datasets using the approaches that were proposed with a combination of all methods are presented in Table 4-9. Using the data gathered from the experiments on all the datasets, we have plotted graphs for cost (2.1) versus the Total Slot conflict in Figure 4.12.



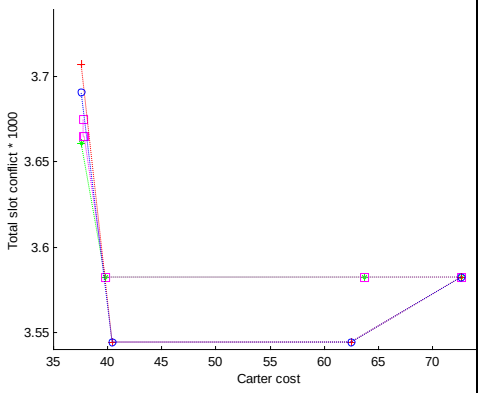
Nott94



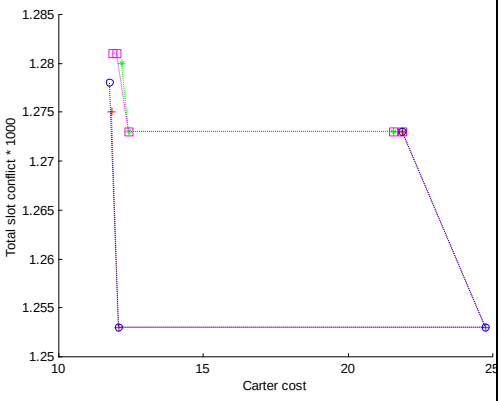
Carf92



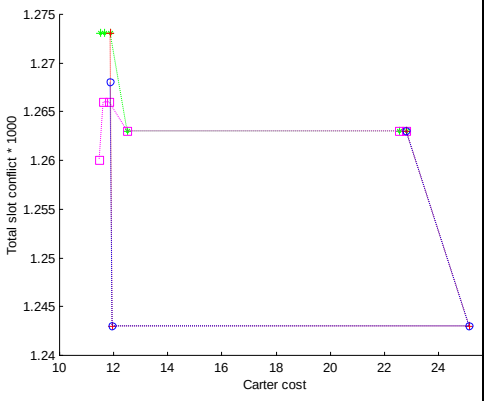
Cars91



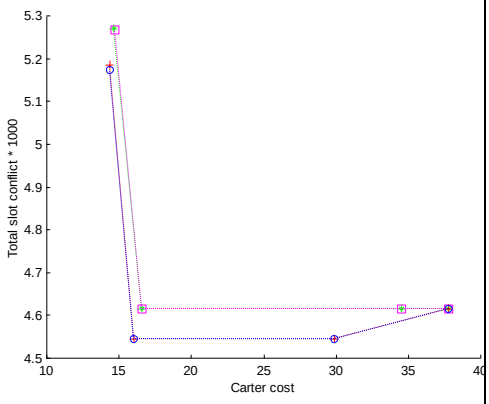
Earf83(a)



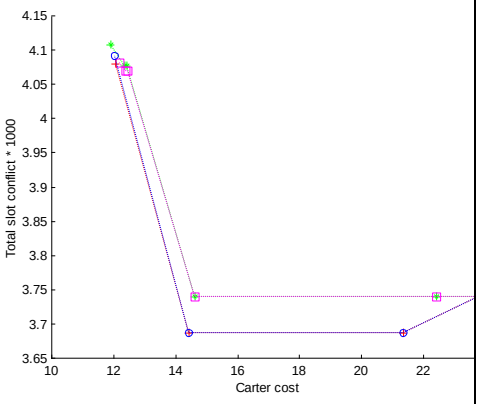
Hecs92(a)



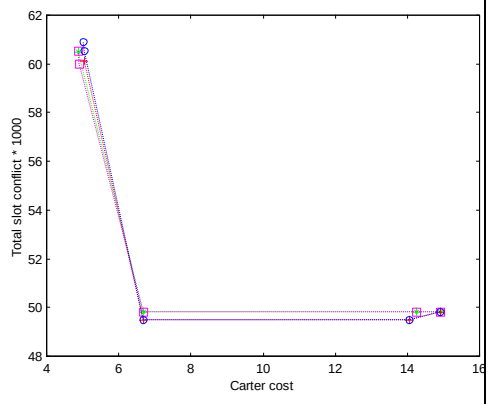
Hecs92(b)



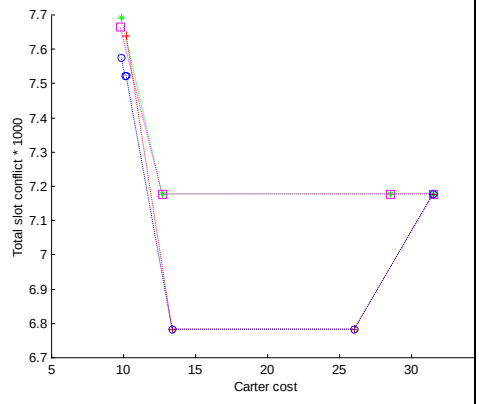
Kfus93



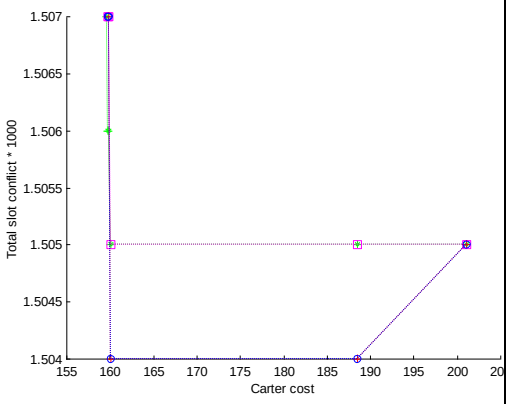
Lsef91



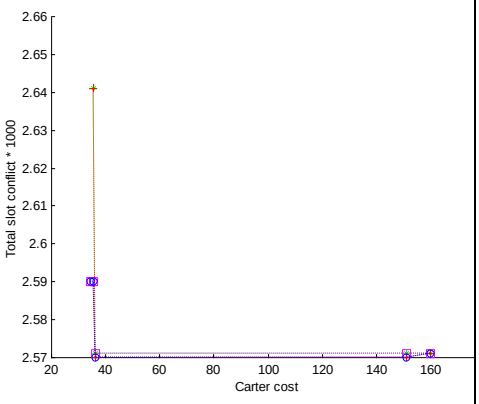
Purs93



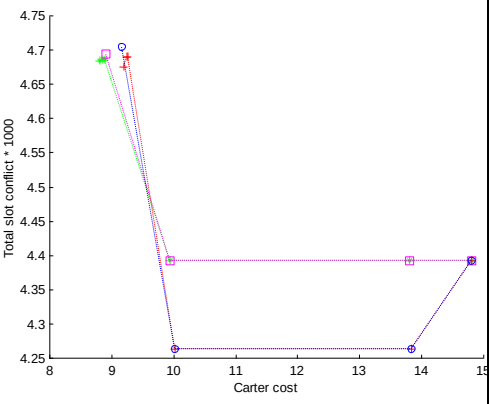
Ryes93



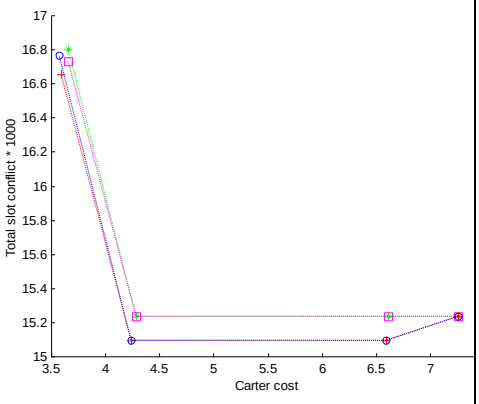
Staf83(a)



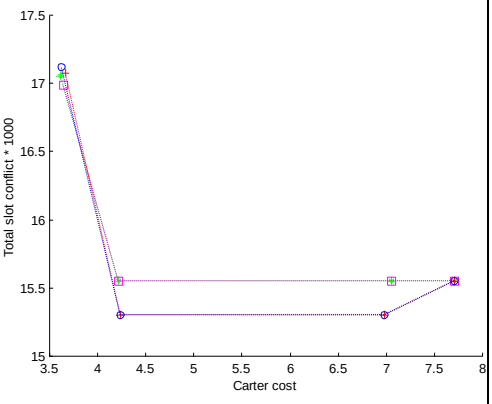
Staf83(c)



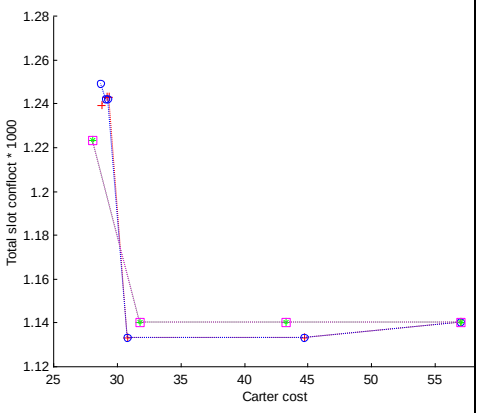
Tres92



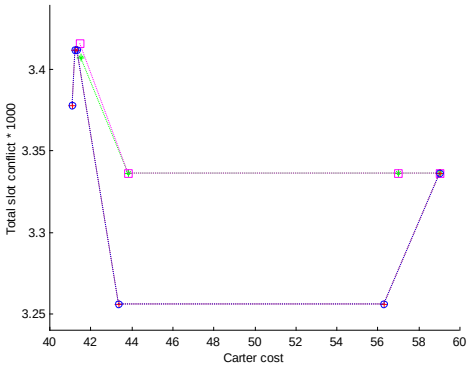
Utas92(a)



Utas92(b)



Utes92



yor-f-83 (I)

Figure 4.12: Graphs for the cost (2.1) versus the Total Slot conflict for all Datasets

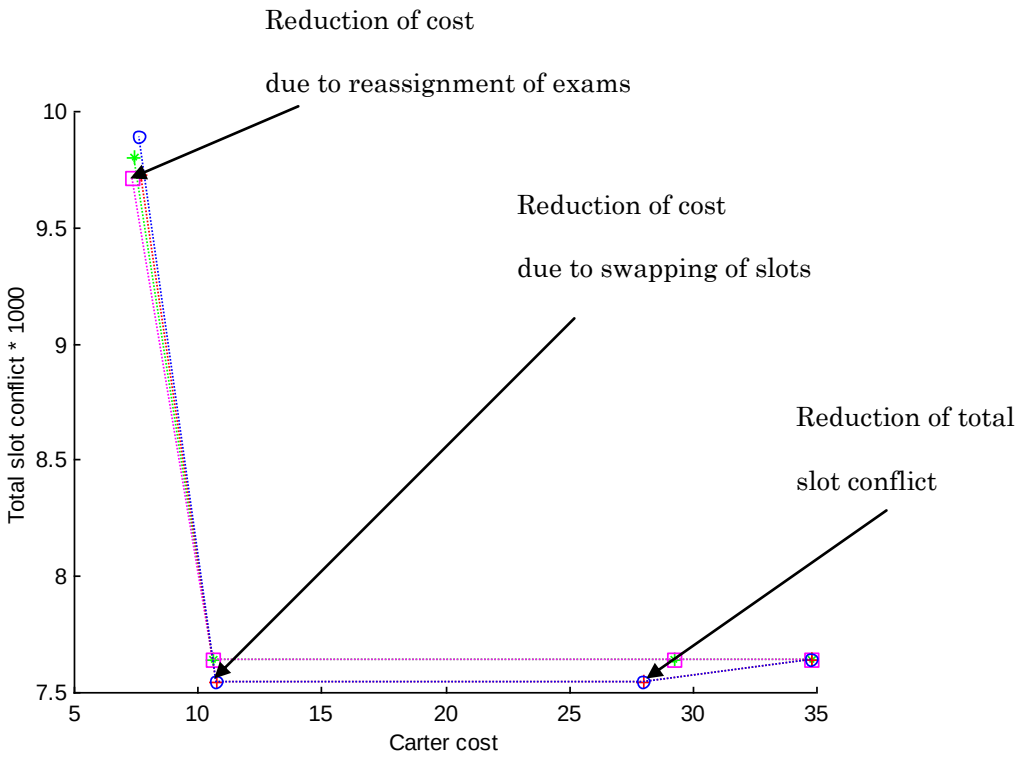


Figure 4.13: General Pattern of Graphs For All Datasets

The pattern generated for all the graphs presented in Figure 4.12 is roughly shown in the above graph, Figure 4.13. In all of these graphs, there actually exist four lines (because they are plotted based on four different combinations of procedures), however, since some of these lines overlap due to similarities in the costs that were obtained for each point, the same trajectory of line is created.

The graph in Figure 4.14 below is an imitation graph that has been created specifically for the purpose of explaining how all the lines exist on the graph. They are created so that all four lines can be easily seen and compared.

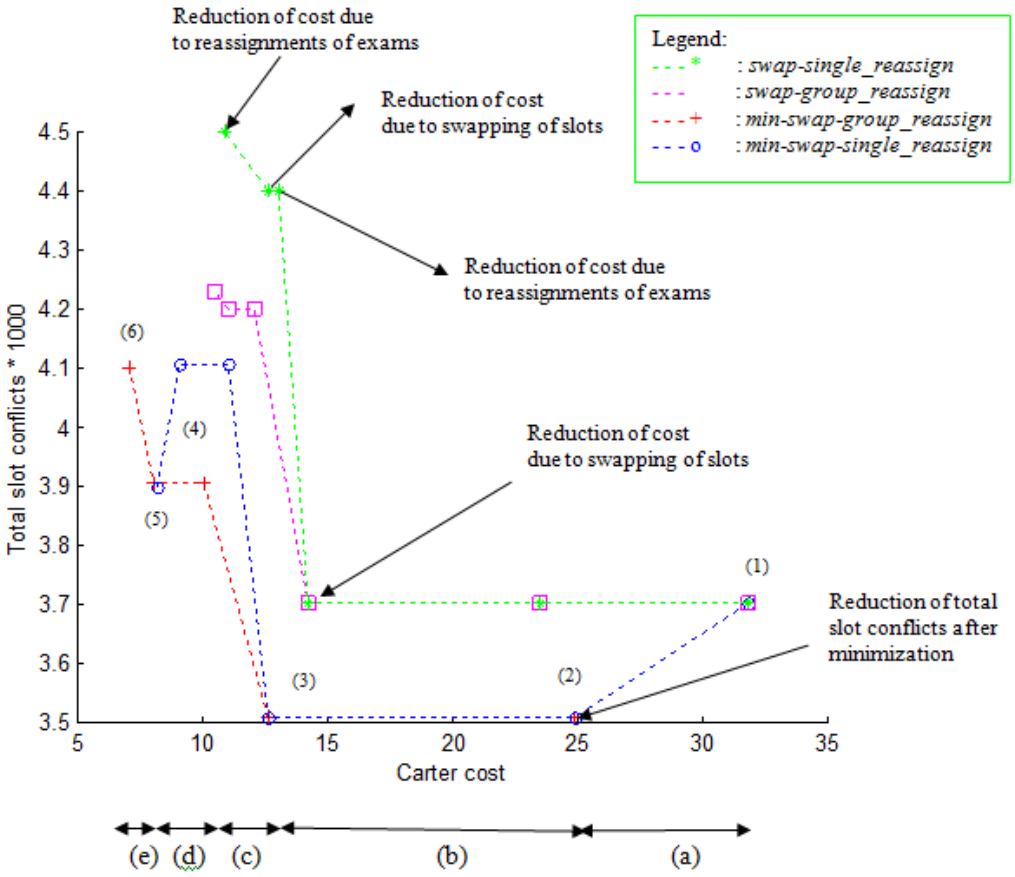


Figure 4.14: Imitation Graph Created For Explanations

The data points (moving from right to left) represented in each graph correspond to the following:

a) First data point (1):

- initial feasible (but not optimal) schedule which was generated via the allocation method

b) Second data point (2) – (if the first trajectory is slanting):

- optimized exam schedule obtained through minimization of total slot conflicts

c) Third data point (3):

- optimized exam schedule obtained through permutation of exam slots

d) Fourth data point (4):

- optimized exam schedule derived through reassignment of exams between slots.

e) Fifth data point (5):

- optimized exam schedule that was arrived at through permutations of exam slots obtained in (d).

f) Sixth data point (6):

- optimized exam schedule obtained through reassignment of exams between slots optimized in (e);

The four lines in each graph are described below:

a) the dotted green line where data points are indicated by asterisks

- after performing both first and second order optimization (permutations of exam slots and single reassignment): known as *swap-single_reassign*

b) the dotted purple line where data points are indicated by squares

- after performing both first and second order optimizations (permutations of exam slots and group reassignment): known as *swap-group_reassign*

c) the dotted blue line where data points are represented by empty circles

- after performing a minimization of total slot conflicts together with both the first and second order optimizations (permutations of exam slots and single reassignment): known as *min-swap-single_reassign*

d) the dotted red line where data points are indicated by the plus sign

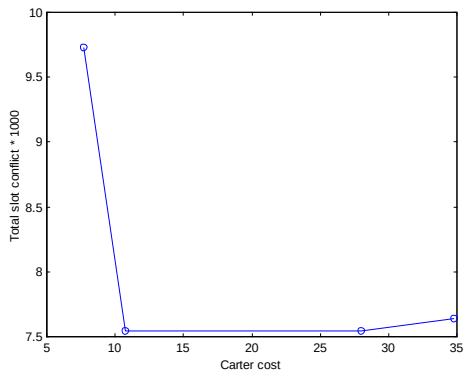
- after performing a minimization of total slot conflicts together with both the first and second order optimizations (permutations of exam slots and group reassignment): known as *min-swap-group_reassign*

4.1.5 Summary of Results and Graphs for Best Cost Produced For Benchmark Datasets

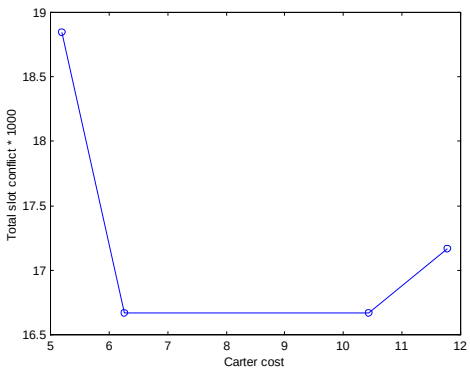
Based on the results presented earlier, the best cost produced for each dataset using the proposed method is summarized in the following table (Table 4-9). It is recalled that, there were four lines in each of the graphs presented before, so the best cost is determined by the line in which it managed to record the best cost. Graphs for each dataset according to the best cost produced are plotted and can be found in Figure 4.15.

Table 4-9: Computational Results (Best Cost) of the Proposed Approach Applied to the Nott and Toronto Dataset

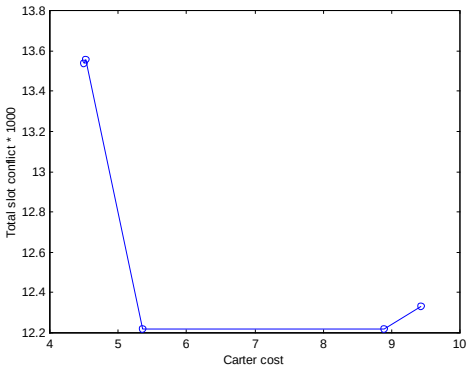
Dataset	No of Slots	Initial Cost	Total Slot Conflicts	Minimiza-tion of Slot Conflicts	Total Slot Conflicts	Current Cost	G/S	Cost After Swap 1	Cost After Reassign I	Total Slot Conflicts	Cost After Swap II	Cost After Re-assign II	Total Slot Conflicts
Nott	23	38.99	8589	YES	8090	31.95	S	10.94	7.34	9979	7.34	7.34	9979
car-s-91 (I)	35	11.77	17169	YES	16665	10.43	S	6.26	5.19	18847	5.19	5.19	18847
car-f-92 (I)	32	9.43	12332	YES	12217	8.89	G	5.36	4.52	13558	4.52	4.49	13535
ear-f-83(I)	24	72.69	3582	YES	3544	62.57	S	40.45	37.57	3707	37.57	37.57	3707
hec-s-92(I)	18	22.83	1263	NO	1263	22.55	G	12.52	11.85	1266	11.62	11.47	1260
kfu-s-93	20	37.79	4616	YES	4544	29.89	G	16.06	14.36	5174	14.36	14.36	5174
lse-f-91	18	23.77	3739	NO	3739	22.42	S	14.63	12.41	4077	12.35	11.90	4107
pur-s-93 (I)	42	14.91	49821	NO	49821	14.27	G	6.69	4.92	60005	4.92	4.88	60532
rye-f-92	23	31.50	7178	NO	7178	28.55	G	12.68	9.80	7664	9.80	9.80	7664
sta-f-83(I)	13	201.95	1507	YES	1505	193.47	G	158.43	158.25	1507	158.25	158.25	1507
tre-s-92	23	14.81	4392	YES	4251	13.25	G	9.84	8.77	4714	8.77	8.74	4719
uta-s-92(I)	35	8.71	15859	YES	15416	8.28	S	4.24	3.59	16792	3.59	3.59	16792
ute-s-92	10	60.71	1200	YES	1149	46.57	G	29.82	27.37	1274	27.37	27.37	1274
yor-f-83 (I)	21	59.04	3336	YES	3256	56.31	G	43.36	41.35	3412	41.27	41.10	3378



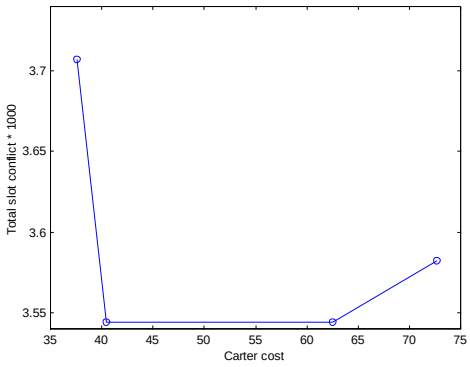
Nott



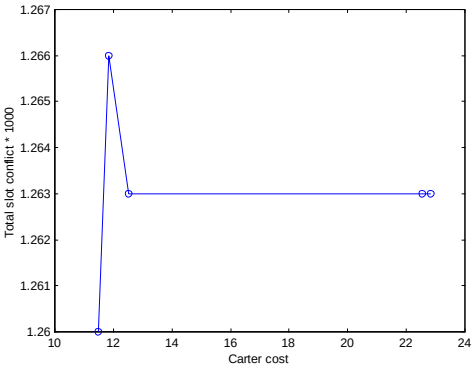
car-s-91 (I)



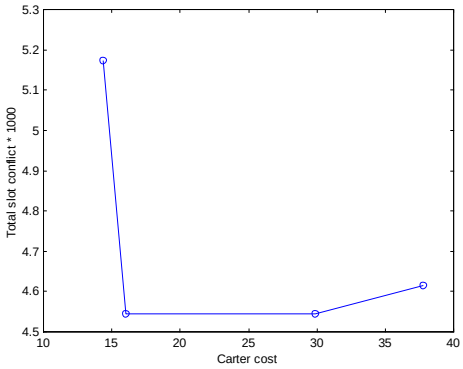
car-f-92 (I)



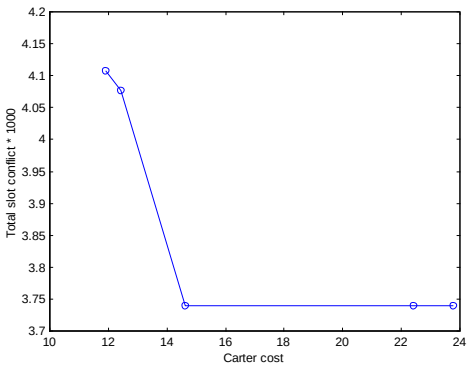
ear-f-83(I)



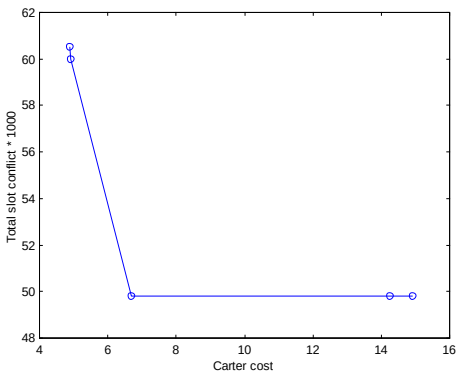
hec-s-92(I)



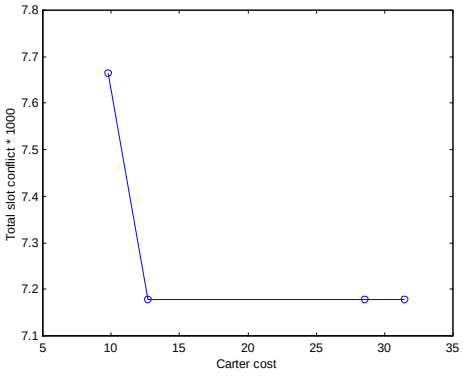
kfu-s-93



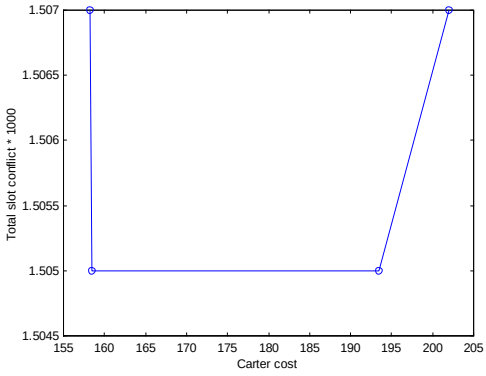
lse-f-91



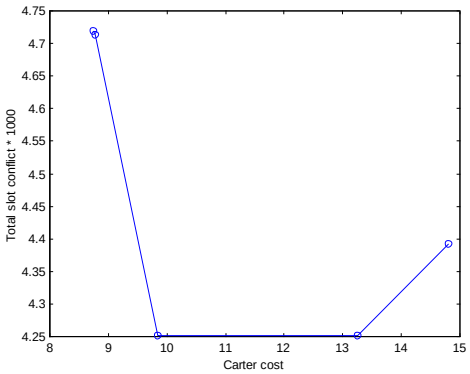
pur-s-93 (I)



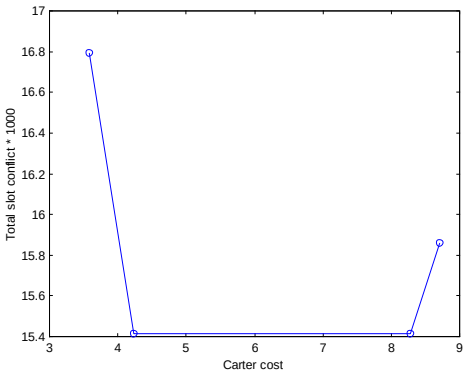
rye-f-92



sta-f-83(I)



tre-s-92



uta-s-92(I)

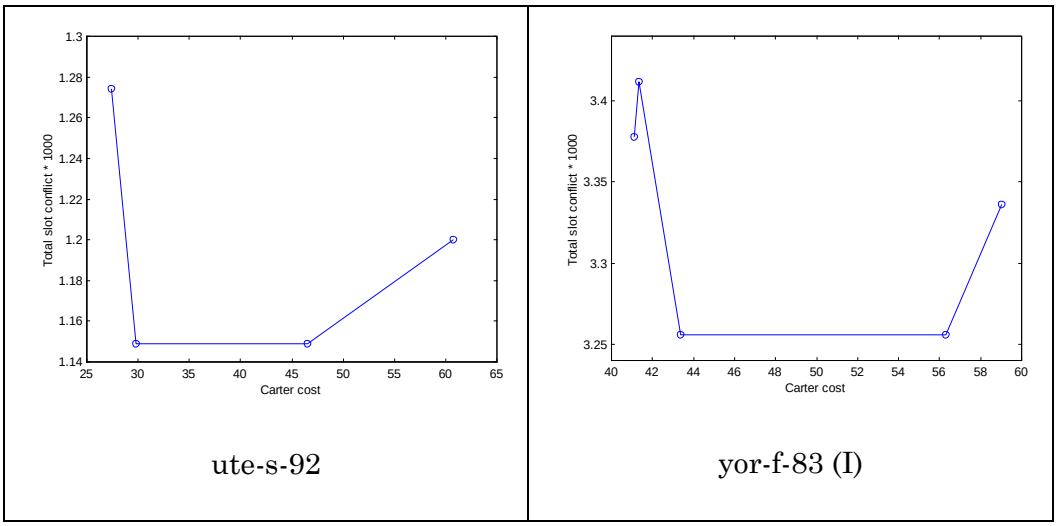


Figure 4.15: Cost (2.1) vs. the Total Slot Conflicts For Nott and Toronto Dataset

4.1.6 Deterministic Pattern Obtained For All Tested Datasets

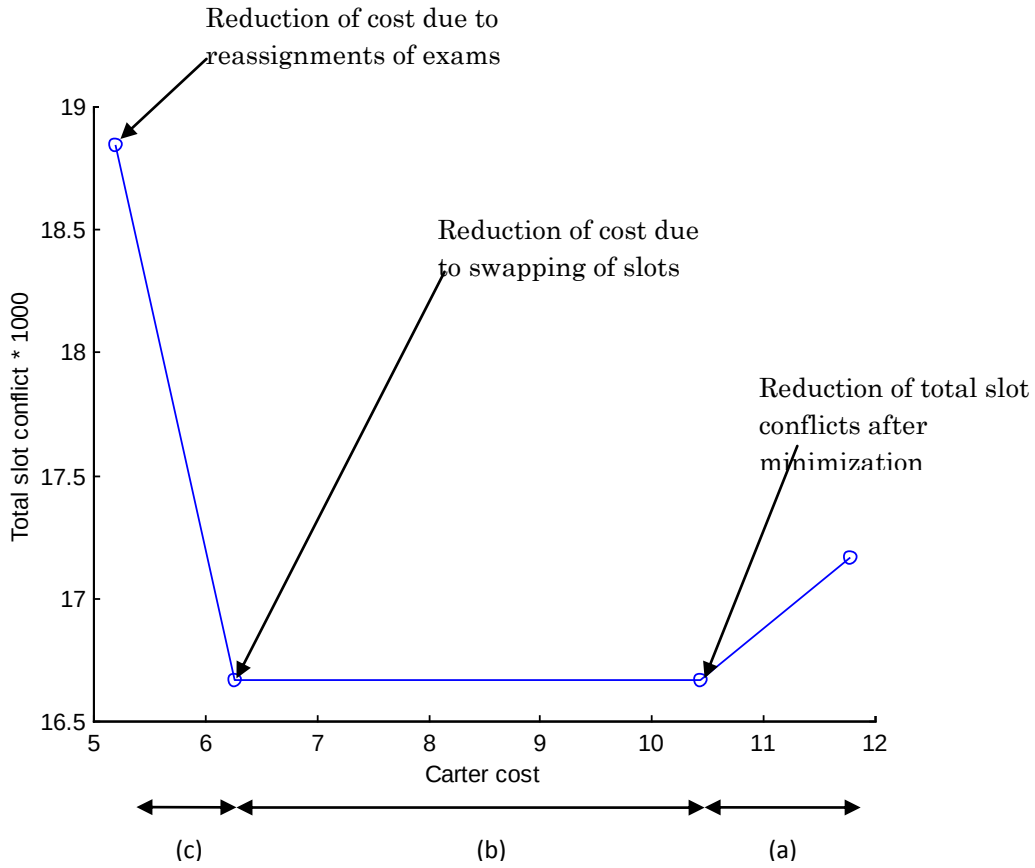


Figure 4.16: The Predicted Pattern of the Graph with the Proposed Approach

When the general pattern of the lines (graphs) are observed it can be concluded that they consist of 3 stages, which can be named as section (a), (b), and (c) from right to left, as illustrated in Figure 4.16. A decrease of the total slot conflicts in section (a) is typically (but not necessarily) coupled with a decrease in the exam schedule cost. In the second stage, in section (b), the exam schedule cost is reduced without any augmentation of the total slot conflicts. The third stage, represented in section (c), resulted in a reassignment of exams that reduced the exam schedule cost but the total slot conflicts increased. However, for some datasets (hec-s-92(I), lse-f-91, pur-s-93(I), and rye-f-92), only section (b) and (c) can be seen on the graph because the best results have already been recorded without running the minimization of slot conflicts procedure.

It is agreed with Lewis (2008), that when certain benchmark datasets are relied upon to evaluate an algorithm, the resulting algorithm could be inclined towards the criteria of the benchmark datasets. Therefore besides the proposed approach being tested on Toronto datasets, other benchmark datasets, as given below, will also be used to prove the universality of the algorithm. These datasets are: Notts and ITC2007.

As mentioned above, in order to test the flexibility of our approach and to ensure that it could work well when applied to other datasets, the methods were tested further on the International Timetabling Competition 2007 (ITC2007) dataset, which can be easily obtained from http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php.

Additional constraints are contained in the ITC2007 dataset including room capacities, period utilization, period related and room related in the objective function. Some important characteristics of the ITC2007 benchmark dataset are presented in Table 4-10. The results obtained for all exams in this dataset can be seen in Table 4-11. The cost (2.1) vs. the Total Slot Conflicts is plotted in Figure 4.17.

Table 4-10: The characteristics of the ITC2007 dataset

Name of Dataset	**No of Exams**	**No of Students**	**Required No of Slots**	**Conflict Density**
Exam1	607	7891	54	5.05
Exam2	870	12743	40	1.17
Exam3	934	16439	36	2.62
Exam4	273	5045	21	15.0
Exam5	1018	9253	42	0.87
Exam6	242	7909	16	6.16
Exam7	1096	14676	80	1.93
Exam8	598	7718	80	4.55

From the results presented in Table 4-9 it is clear that the optimization of the initial feasible timetable resulted in an improved exam timetable with a lower cost (2.1). For the "Nott" dataset, a reduction of the cost from 38.99 to 10.94 was obtained after the permutations of exam slots on the initial schedule. The cost was further improved to 7.34 after the reassignment of exams.

For the ITC2007 dataset, significant reductions of the exam schedule cost are also shown in the results reported in Table 4-11 when compared to the cost of the original feasible schedule. For example, the

cost of the exam schedule evaluated against the benchmark problem Exam8 in the ITC2007 dataset was reduced from 25.15 to 0.32 by the permutations of exam slots and was further improved to 0.14 by the reassignment of exams. It is worth noting that for this benchmark problem a second round of slot swapping and exam reassignments resulted in further improvement to the cost from 0.14 to 0.13. However, for most benchmark problems a single round of optimization was sufficient to achieve a competitive exam schedule that could not be improved upon in the second round.

Table 4-11: Computational Results of the Proposed Approach Applied to the ITC2007 Dataset

Dataset	No of Slots	Initial Cost	Total Slot Conflicts	Minimiza-tion of Slot Conflicts	Total Slot Conflicts	Current Cost	G/S	Cost After Swap 1	Cost After Reassign I	Total Slot Conflicts	Cost After Swap II	Cost After Re-assign II	Total Slot Conflicts
Exam1	54	23.90	7522	YES	7414	23.49	G	2.02	1.12	10787	1.12	1.12	10787
Exam2	40	26.92	4740	YES	4709	26.92	G	0.48	0.22	5359	0.22	0.22	5359
Exam3	36	28.53	9114	YES	8928	28.53	G	3.35	1.84	12584	1.84	1.84	12584
Exam4	21	33.84	4001	YES	3958	28.49	G	14.62	12.06	4326	12.06	12.06	4326
Exam5	42	41.79	5156	YES	5118	41.79	G	0.83	0.37	5736	0.37	0.37	5736
Exam6	16	13.32	1652	YES	1647	13.32	G	5.50	4.70	1960	4.69	4.61	1954
Exam7	80	23.38	9949	YES	9839	23.55	G	0.16	0.07	11066	0.07	0.07	11066
Exam8	80	25.15	6843	YES	6706	25.15	G	0.32	0.14	7374	0.13	0.13	7374

An important feature of the proposed optimization is its deterministic pattern that is preserved for all the datasets. The minimization of the total slot conflicts has proven to be a useful preparatory step for the subsequent minimization of the cost of the exam schedule. Where the slot conflicts were minimized the greatest "packing" of conflicting exams was achieved and, by doing so, the possibility of reducing the schedule cost in subsequent steps was enhanced. It should be noted that this is beneficial even if, in some rare circumstances (see Exam7 in the ITC2007 dataset, Table 4-11; cost increase from 23.38 to 23.55) the reduction of the total slot conflicts comes at the expense of some increase to the schedule cost. This enhanced potential for subsequent reduction to the schedule cost is fully capitalized on in the subsequent step of permutation of exam slots; where the cost was reduced to 0.07.

The permutation of exams slots is a very simple approach and yet it produces a very significant reduction of the cost (2.1) of the initial exam schedule. By splitting the exam scheduling problem into three sub-problems of minimization of slot conflict, minimization of cost by slot swapping, and minimization of cost by reassignments we achieved a clear deterministic progression of the optimization process that lends itself to easy interpretation.

The reassignments of exams also never fail to reduce the cost (2.1). The details showed that group reassignments outperformed single reassignments in most of the datasets. The effect of these reassignments can be seen from the third data point to the fourth data point in each line in the graphs given. There is a very clear pattern, whereby, for each line,

the graph is seen to rise on a diagonal to the left. This indicated that the exam schedule generated at this stage has a lower cost but an increase in the overall total slot conflicts.

While the single reassignment follows the strict minimization of the cost (2.1), with group reassignment the benefit comes from the inherent interaction of the effects of reassignment of exams in a group. Although the individual exams in a group have been selected according to their potential to reduce the cost (2.1), when reassigned to another slot, taken together with other exams in a group, this potential for a reduction in cost (2.1) may be weakened or reversed. Although this is unwelcome, it allows the search to escape from local optima and thus improve on the single reassignment solution. An alternative strategy might be to perform a different type of optimization with a single reassignment that would allow the search to escape from local optima (e.g. simulated annealing) but it is recommend that the benefits be weighed against the computational cost before an approach is proceeded with.

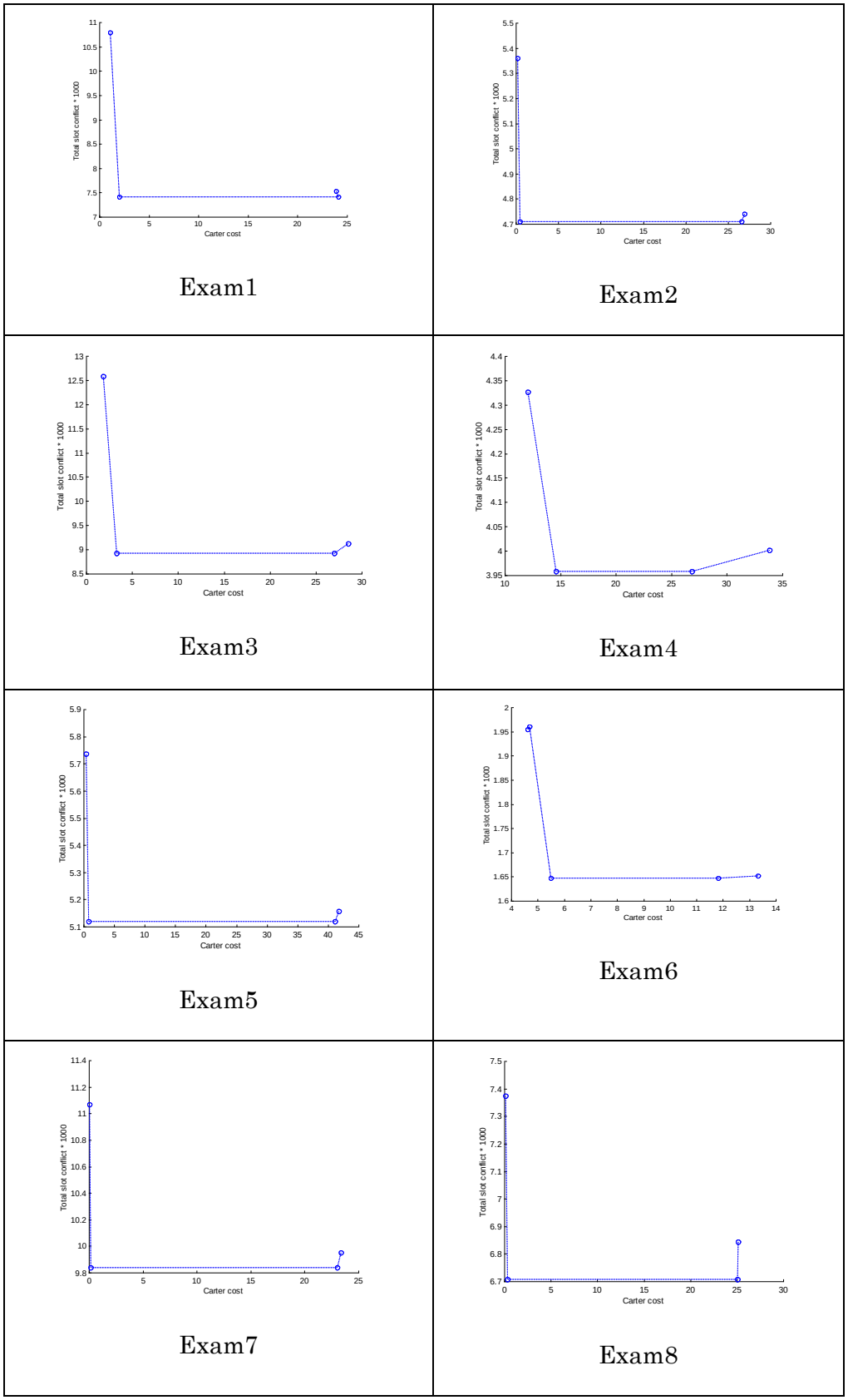


Figure 4.17: Cost (2.1) vs. the Total Slot Conflicts for ITC2007 Dataset

Additionally, the feasible solutions with the lower total slot conflicts appear to offer an advantage in terms of their increased capacity to minimize the cost (2.1) through simple re-ordering of the slots and, subsequently, through the re-assignment of exams between slots. However, while at the initial stages of optimization one is justified in making a positive correlation between the cost and the slot conflicts count (as is endorsed by the experiences of other researchers using max-degree pre-ordering of exams in their scheduling heuristics), it is clear that this correlation represents a potential for the reduction in cost by swapping the slots. At the final stages of optimization, this potential is not relevant, as the slots are deemed to have been optimally ordered already. In the rare circumstance where the reassignment of exams was discovered to create an opportunity for further cost reduction by swapping the exam slots, a second round of optimization delivered the expected improvement of the exam schedule.

4.1.7 Comparison of the Proposed Methods Compared to Other Constructive Methods in the Literature

A comparison of our results in terms of Carter cost (2.1) with the results obtained from other constructive methods reported in the literature is presented in Table 4-12. We have decided to analyse the results in a statistical approach by evaluating the distance between the Carter cost (2.1) obtained from a particular method against the best Carter cost (2.1) obtained in the literature. In this context, a mean percentage discrepancy between the solution delivered by a given method and the best solution reported in the literature, together with the standard deviation of such discrepancies, evaluated on a representative

set of benchmark problems, provide a measure of reliability of the examination scheduling method. In particular, the low value of the standard deviation indicates that the method consistently delivers good results. In Table 4-12 the mean shown at the bottom of each column is the mean calculated for available solutions obtained for each method in this table, however they are not conclusive because there are some methods that could not produce a feasible solution for a particular dataset. Hence we will analyze the data by omitting the datasets that could not be solved by all methods. We present the results in Table 4-13. We observe that the mean (average percentage difference) obtained by our approach is 5.02 is at the 6th position however with the standard deviation of 2.8 (2nd position) shows that the approach is predictable is terms of performance. Looking at Table 4-12 and Table 4-13, it is clear that the methods that are listed have a rather uneven performance. They perform well on some benchmark problems and not as well on others. In addition to that, some approaches even fail to produce any feasible solution. This is a rather unwelcome characteristic from the user's perspective, as there is no way of predicting the quality of the solution that will be obtained using a particular method on a new dataset.

The goal of our research is to propose a general algorithm that will provide a solution for all existing datasets or new ones. In relation to that we will be analyzing the results by omitting methods that are unable to produce results in any one of the datasets. Table 4-14 shows that our proposed method is very competitive, with a mean percentage discrepancy of 9.11% between its solutions and the best known ones, and is by far the

most consistently reliable, as indicated by the standard deviation 9.77 of these discrepancies.

One important point that should be noted when comparing the performance of the various methods is that several of the best results have been obtained by using methods that did not report any results for pur-f-93 and/or rye-f-92. This is significant because the quality, consistency, and the universal applicability of the method are highlighted.

The proposed optimization approach is a very simple yet very competitive one in generating reliably high quality exam schedules. It is believed that the domain transformation approach, that facilitated the transformation of a complex optimization problem into a sequence of more tractable optimizations, has the potential for successful application in a broader spectrum of applications. An important feature of the optimization method is that, the feasibility of the initial solution throughout the whole of the optimization is preserved, thus saving a considerable computational effort compared to other methods that require customized post-processing.

Table 4-12: Results in Terms of Carter cost (2.1) of Our Method in Comparison with Some Other Constructive Methods in the Literature

(Highlighted columns are for the methods that delivered results for all instances in the Toronto dataset.)

Dataset	**GH**	**AHO**	**ADC**	**MHO**	**OH**	**LCH**	**ADO**	**ALC**	**Ours**
car-s-91 (I)	7.1	**4.97**	5.45	5.29	5.08	5.03	5.17	5.12	5.19
car-f-92 (I)	6.2	4.32	4.5	4.54	4.38	**4.22**	4.74	4.41	4.49
ear-f-83(I)	36.4	36.16	36.15	37.02	38.44	**36.06**	40.91	36.91	37.57
hec-s-92(I)	**10.8**	11.61	11.38	11.78	11.61	11.71	12.26	11.31	11.47
kfu-s-93	**14**	15.02	14.74	15.8	14.67	16.02	15.85	14.75	14.36
lse-f-91	**10.5**	10.96	10.85	12.09	11.69	11.15	12.58	11.41	11.9
pur-s-93 (I)	**3.9**	-	-	-	-	-	5.87	5.87	4.88
rye-f-92	**7.3**	-	-	10.38	9.49	9.42	10.11	9.61	9.8
sta-f-83(I)	161.5	161.9	**157.21**	160.4	157.72	158.86	158.12	157.52	158.25
tre-s-92	9.6	8.38	8.79	8.67	8.78	**8.37**	9.3	8.76	8.74
uta-s-92(I)	3.5	**3.36**	3.55	3.57	3.55	3.37	3.65	3.54	3.59
ute-s-92	**25.8**	27.41	26.68	28.07	26.63	27.99	27.71	26.25	27.37
yor-f-83 (I)	41.7	40.88	42.2	39.8	40.45	**39.53**	43.98	39.67	41.1
Mean of Carter cost*	26.02	29.54	29.23	28.12	27.71	27.64	26.94	25.78	26.05

**Mean of Carter cost (2.1) for datasets obtained by each method*

GH-Graph Heuristics- (Carter and Laporte, 1996), AHO-Adaptation of Heuristics Orderings-(Burke, E. K. and Newall, J. P., 2004a), ADC-Adaptive Decomposition and Construction- (Qu, R. and Burke, E. K., 2007), MHO-Multiple Heuristics Orderings-(Asmuni et al., 2009), OH-Ordering Heuristics-(Abdul-Rahman et al., 2009), LCH-Linear Combinations of Heuristics-(Burke et al, 2010c), ADO-Adaptive Decomposition and Ordering- (Abdul-Rahman et al., 2011), ALC-Adaptive Linear Combination-(Abdul-Rahman et al., 2014).

Table 4-13: Average Percentage Distance to the Optimal Cost for 11 Datasets in the Toronto Problem

Dataset	**GH**		**AHO**		**ADC**		**MHO**		**OH**		**LCH**		**ADO**		**ALC**		**Our Proposed Method**		**Best Cost****
	Cost	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	**Cost**	**%**	
car-s-91 (I)	7.1	42.86	**4.97**	0.00	5.45	9.66	5.29	6.44	5.08	2.21	5.03	1.21	5.17	4.02	5.12	3.02	5.19	4.24	**4.97**
car-f-92 (I)	6.2	46.92	4.32	2.37	4.5	6.64	4.54	7.58	4.38	3.79	**4.22**	0.00	4.74	12.32	4.41	4.50	4.49	6.01	**4.22**
ear-f-83(I)	36.4	0.94	36.16	0.28	36.15	0.25	37.02	2.66	38.44	6.60	**36.06**	0.00	40.91	13.45	36.91	2.36	37.57	4.02	**36.06**
hec-s-92(I)	**10.8**	0.00	11.61	7.50	11.38	5.37	11.78	9.07	11.61	7.50	11.71	8.43	12.26	13.52	11.31	4.72	11.47	5.84	**10.8**
kfu-s-93	**14**	0.00	15.02	7.29	14.74	5.29	15.8	12.86	14.67	4.79	16.02	14.43	15.85	13.21	14.75	5.36	14.36	2.51	**14**
lse-f-91	**10.5**	0.00	10.96	4.38	10.85	3.33	12.09	15.14	11.69	11.33	11.15	6.19	12.58	19.81	11.41	8.67	11.9	11.76	**10.5**
sta-f-83(I)	161.5	2.73	161.9	2.98	**157.21**	0.00	160.4	2.03	157.72	0.32	158.86	1.05	158.12	0.58	157.52	0.20	158.25	0.66	**157.2**
tre-s-92	9.6	14.70	8.38	0.12	8.79	5.02	8.67	3.58	8.78	4.90	**8.37**	0.00	9.3	11.11	8.76	4.66	8.74	4.23	**8.37**
uta-s-92(I)	3.5	4.17	**3.36**	0.00	3.55	5.65	3.57	6.25	3.55	5.65	3.37	0.30	3.65	8.63	3.54	5.36	3.59	6.41	**3.36**
ute-s-92	**25.8**	0.00	27.41	6.24	26.68	3.41	28.07	8.80	26.63	3.22	27.99	8.49	27.71	7.40	26.25	1.74	27.37	5.74	**25.8**
yor-f-83 (I)	41.7	5.49	40.88	3.42	42.2	6.75	39.8	0.68	40.45	2.33	**39.53**	0.00	43.98	11.26	39.67	0.35	41.1	3.82	**39.53**
Average Percentage Difference*	**mean=10.71 std=17.46**		**mean=3.14 std=2.92**		**mean=4.67 std=2.82**		**mean=6.83 std=4.52**		**mean=4.79 std=3.01**		**mean=3.64 std=4.96**		**mean=10.48 std=5.19**		**mean=3.72 std=2.50**		**mean=5.02 std=2.80**		

**Average Percentage Difference To The Best Constructive Carter Cost(%) in the Literature*

*** Best Constructive Carter Cost (2.1) Reported in the Literature*

GH-Graph Heuristics- (Carter and Laporte, 1996), AHO-Adaptation of Heuristics Orderings-(Burke, E. K. and Newall, J. P., 2004a), ADC-Adaptive Decomposition and Construction- (Qu, R. and Burke, E. K., 2007), MHO-Multiple Heuristics Orderings-(Asmuni et al., 2009), OH-Ordering Heuristics-(Abdul-Rahman et al., 2009), LCH-Linear Combinations of Heuristics-(Burke et al, 2010c), ADO-Adaptive Decomposition and Ordering- (Abdul-Rahman et al., 2011), ALC-Adaptive Linear Combination- (Abdul-Rahman et al., 2014).

Table 4-14: Average Percentage Distance to the Optimal Cost

Dataset	[12]		[22]		[23]		Our Proposed Method		Best Constructive Cost
	Cost	%	Cost	%	Cost	%	Cost	%	
car-s-91 (I)	7.10	42.86	5.17	4.02	5.12	3.02	5.19	4.43	4.97
car-f-92 (I)	6.20	46.92	4.74	12.32	4.41	4.50	4.49	6.40	4.22
ear-f-83(I)	36.40	0.94	40.91	13.45	36.91	2.36	37.57	4.19	36.06
hec-s-92(I)	10.80	0.00	12.26	13.52	11.31	4.72	11.47	6.20	10.80
kfu-s-93	14.00	0.00	15.85	13.21	14.75	5.36	14.36	2.57	14.00
lse-f-91	10.50	0.00	12.58	19.81	11.41	8.67	11.90	13.33	10.50
pur-s-93 (I)	3.90	0.00	5.87	50.51	5.87	50.51	4.88	25.13	3.90
rye-f-92	7.30	0.00	10.11	38.49	9.61	31.64	9.80	34.25	7.30
sta-f-83(I)	161.50	2.73	158.12	0.58	157.52	0.20	158.25	0.66	157.21
tre-s-92	9.60	14.70	9.30	11.11	8.76	4.66	8.74	4.42	8.37
uta-s-92(I)	3.50	4.17	3.65	8.63	3.54	5.36	3.59	6.85	3.36
ute-s-92	25.80	0.00	27.71	7.40	26.25	1.74	27.37	6.09	25.80
yor-f-83 (I)	41.70	5.49	43.98	11.26	39.67	0.35	41.10	3.97	39.53
Average Percentage Difference To Best Constructive cost (%)	**mean = 9.06 std = 16.44**		**mean = 15.72 std = 13.84**		**mean = 9.47 std = 14.72**		**mean = 9.11 std = 9.77**		

The proposed method is also very reliable and stable in producing schedules for larger problem instances, for example, pur-s-93 in the Toronto dataset and Exam7 in the ITC2007 dataset.

It is expected that the proposed method be adapted in a relatively straightforward manner to the capacitated scheduling problem by introducing appropriate granular data structures that will permit the required domain transformation in the optimization process. Also, other constraints, suggested at the 2nd International Timetabling Competition in 2007-08, should fit into the general framework of the proposed method.

The approach proposed in this study to solve the examination scheduling problems is very efficient and reliable in producing good quality examination timetables. It has consistently produced encouraging results for all benchmark datasets, which is not the case for some other constructive methods in the literature. They perform well on some benchmark problems and not as well on others, and in a few cases some methods failed to produce a solution. This is a rather unwelcome characteristic from the user's perspective, as there is no way of predicting the quality of the solution that will be obtained using a particular method on a new dataset. Since the proposed approach produces consistent results when tested on different benchmark datasets, it should be stated that the method is very flexible and highlights the quality, consistency, and potential for universal application. The deterministic optimization pattern achieved on all benchmark datasets, which was consistently maintained, identifies the approach that is proposed as a novel contribution to this area.

CHAPTER 5

Global Search Procedure: Incorporation into the Proposed Optimization Framework

Optimization can be understood simply as a process of improving a set of values in a direction that is desired. In computing, we can imagine optimization as selecting the best set of values from an available set of choices. In examination scheduling, optimization refers to minimizing the cost of the schedule. We have previously demonstrated a few methods proposed for optimization which produced very encouraging results. In this chapter, a study on the effectiveness of incorporating a global search procedure (Genetic Algorithm) into the proposed optimization framework will be made in comparison to our previous incorporation of local search procedure.

5.1 Substitution of a Global Search Procedure in the Optimization Stage of the Proposed Framework

A feasible timetable can have exam orderings which do not satisfy many of the soft constraints. Consequently, a separate optimization process needs

to be deployed to obtain better quality schedules. In Chapter 4, the proposed approach to solving the examination scheduling problem which consists of several stages of optimization was discussed. The phases of the examination scheduling method are as follows:

1. Problem domain transformation from student-exam to exam conflict and spread matrix data space.
2. Generation of a feasible solution via an allocation method and backtracking.
3. Minimization of the overall slot conflicts.
4. Minimization of the schedule cost by slot swapping.
5. Minimization of the schedule cost by exam reallocation.
6. Repetition of the last two steps until no further improvement in the schedule cost can be made.

In the scheduling steps outlined above, optimization is started at the third bullet point and continues until the last bullet point. It should be noted that, for the fourth bullet point, (i.e. *minimization of the schedule cost by slot swapping*), a simple Greedy Hill Climbing (GHC) was introduced. Realizing that the method that was employed (permutation of exam slots), is a local search procedure, it was thought that a global search procedure should be incorporated in order to see whether or not better quality schedules could be generated. For this purpose, we have implemented the Genetic Algorithm (GA) to substitute the permutations of exam slots in the optimization process (Rahim *et al.,* 2013a).

Even though GA does not offer any guarantees with regard to convergence to global optimum, recent research has moved on to explore meta-heuristics based approaches (e.g. harmony search, particle swarm optimization, bee colony optimization, etc.) to enhance the effectiveness of such optimisation and has demonstrated some success. (Burke *et al.*, 1994a; Burke *et al.*, 1994b; Gyori *et al.*, 2001; Ulker *et al.*, 2007). Additionally, it has been confirmed that hybridizations of GA with some local search have led to some success in this area (Qu *et al.*, 2009a). We have therefore chosen GA as an alternative approach to our optimisation (Rahim *et al.,* 2013a).

Below we have presented a diagram (Figure 5.1) to illustrate a summary of the work done in our research which shows the sequence of every process involved and the part that will be substituted by GA. Note that the whole set of optimizations was performed twice, therefore, the first and second order optimizations can be seen in the diagram.

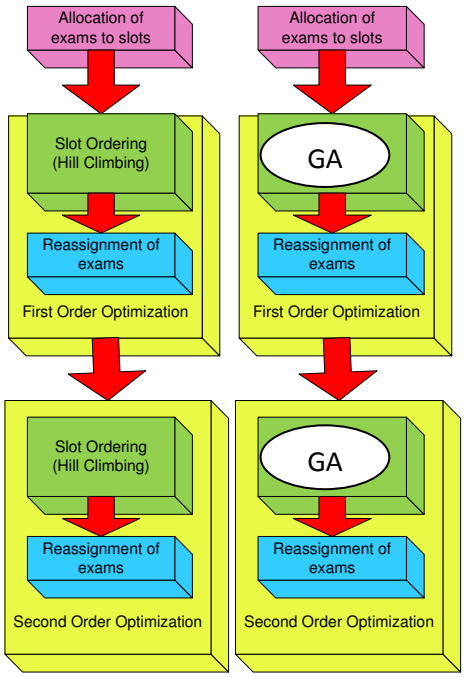


Figure 5.1: Scheduling and Optimization Steps Before and After GA Substitution.

As previously mentioned, the HC optimization was performed by the permutations of the rows/columns of the spread matrix of a feasible schedule obtained via the allocation method. The permutation method should be considered quite an efficient procedure because the number of available exam slots is frequently quite small. In our hypothesis, we postulate that the effect of slot swapping is that new exam ordering will be generated with better quality.

Besides HC, implementation of GA is with the same objective as HC, which is to improve the ordering of the exams in the feasible exam schedule that is generated. The exam slots of the parents involved during crossover at certain points will be randomly exchanged by GA operators, it is therefore suggested that slot swapping to the original slot order from the original feasible schedules be performed (but note that it is a different type of swap compared to the above Hill Climbing).

The main objectives of this research are to study the effectiveness of GA in comparison to HC and to find the best range of parameters (the population size and number of iterations in GA) in the optimization stage of the proposed framework.

5.1.1 Genetic Algorithm

Genetic Algorithm (GA) is a search heuristic that imitates the process of natural evolution. This heuristic is normally used to generate solutions (which are normally good or useful) to optimization and search problems. In a Genetic Algorithm, a population of candidate solutions will evolve towards better solutions. Each individual in the population has some

characteristics (known as chromosomes) that can be mutated and modified.

The evolution process begins from a population of randomly generated individuals and is an iterative process, whereby the population in each iteration is called a generation. In each generation, the fitness (quality) of every individual in the population is evaluated based on the objective function. Good quality individuals (normally the two best individuals) are selected from the current population, and each individual's genome (characteristics) is modified (recombined through crossovers and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Normally, the algorithm terminates when either a maximum number of generations has been produced or a superior solution has been obtained for the population.

The effectiveness of GA highly depends on the tweaks that are made to its parameters. Despite the simplicity of its algorithm, GA requires careful and intelligent settings to be given to its parameters, for example: the method of selecting parents, the population size, and the crossover type and rate; mutation type and rate, the number of iterations etc. An optimal calibration of the parameters might cause the algorithm to converge on the best results in an efficient time, meanwhile, on the other hand, non-optimal configurations of the parameters might cause it to take a longer time to produce good solutions and, in some cases, good solutions may not be obtained at all.

5.1.2 Our Genetic Algorithm Implementation

In our Genetic Algorithm (GA) implementation (Rahim *et al.,* 2013a; Rahim *et al.,* 2013c), we defined the original parent *P0*, which is a data structure with the initial ordering of slots *(1 …. N)* where *N* is the number of slots. GA creates a new parent by moving the position of the rows in blocks from a random index *K* to the end of *P0* to generate the first *K-N* number of rows. To complete the parent, the balance (first row to *K*) is then taken from *P0* from position 0 to K to be filled in at position *K + 1* to *N* in the new parent. A number of *npar* parents will be created. In general, the generation of the new parents is created by shifting the rows that, in the end, are the new representation of the original parent with a magnitude maximum distance of *npar* – 1. Therefore, if it is just a window shift, there will be identical parents, especially when the number of *N* is smaller than the number of *npar*.

We then produced the new offsprings from these initial parents. The number of offsprings to be produced is equal to the unique permutation of a parent in *P* with other parents in *P*. Each of the parents will be crossed over with all other parents at a certain random point *R*, creating two new offsprings. *Offspring1* will contain the first to *R* slot from *Parent1* and will then be completed by appending the slot in *Parent2* starting from the first to *N* which is not already in *Offspring1*. The same goes for *Offspring2*. It will contain the first to *R* slot from *Parent2* and will be completed by appending the slot in *Parent1*. These two newly created offsprings will be added to "*O*" which is the overall population. Any identical offsprings will be eliminated and replaced with a mutated *P* where we interchange a random slot *t* with a random slot *u*. The best

offspring with the lowest cost will be automatically selected to become the next generation parent *P*.

The offsprings generated in the population are the individuals with a new ordering of slots due to the crossover of the parents involved, which, in this study, is assumed to be a type of slot swapping. The process of slot swapping performed using GA is illustrated in Figure 5.2 and Figure 5.3. In the process of generating the offspring there is a possibility of having a redundant slot that has been obtained from the first parent when the slots from the second parent is moved to the new child. We alleviate this by substituting the identical slot with a slot that was not in the solution by replacing the missing slot at the location of the redundant slot. In Figure 5.3 slot number 7 has been selected from the first parent and another selected from the second parent. The second occurrence of slot 7 is being replaced by slot 2 which is missing in the new offspring.

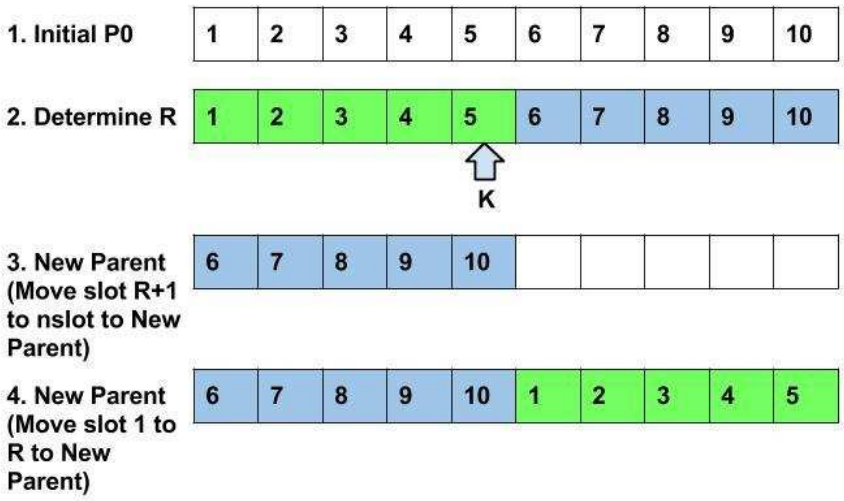


Figure 5.2: Generation of New Parents in the Proposed GA

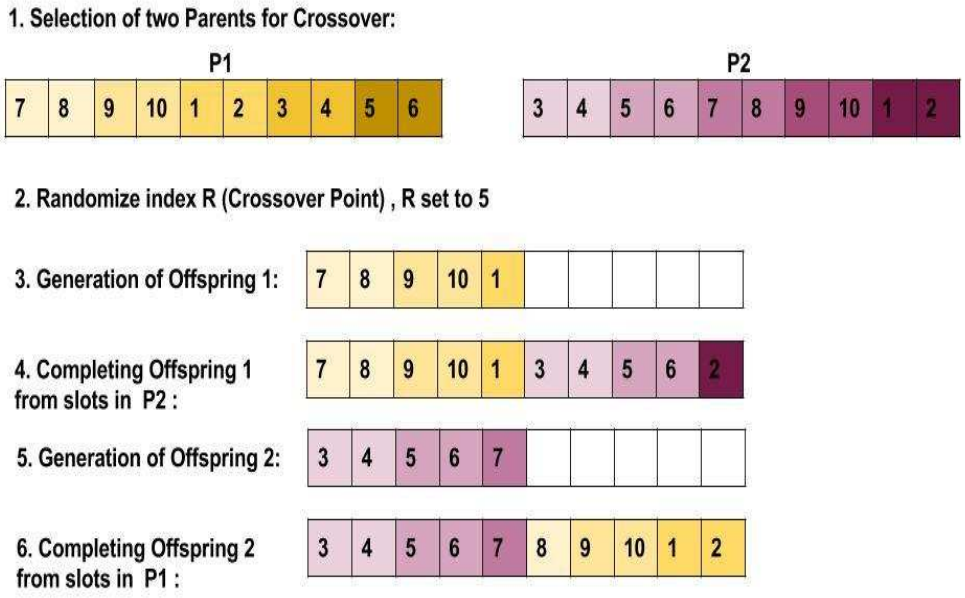


Figure 5.3: Generation of Offsprings in the Proposed GA

Generating parents and offsprings via this technique does not introduce any non-feasible solution as exams in all slots remain in their slot, thus it does not create any new conflicts that will create unfeasible solution. However the change due to slot shift will affect the cost function value as the change shift the distance between exams (contributing factor to Carter cost (2.1)) that are dependent among the slots. Recollect that each of these slots has exams that are not in-conflict and each of these exams contains students. The beauty of granularization is that we no longer need to work at the micro level but we can solve the problem at a higher perspective of the problem space. The proposed GA approach has the whole search space of feasible solutions to work on unlike other GA approaches, where the GA has the whole search space which includes infeasible solutions. Having only feasible solutions in the search space makes the approach efficient and fast. The proposed GA was tested on the

Toronto dataset and the results / performance will be discussed in the next section.

5.1.3 Results for Hill Climbing versus Genetic Algorithm Optimization

In order to determine the effectiveness of the proposed GA in the optimization step for the examination scheduling, experiments on the Toronto benchmark repository have been conducted. Furthermore, all nine combinations of parameters for different number of parents and iterations (i.e.: parents: 10 / 15 / 20 and iterations 10 / 15 / 20) were used during the experiments.

As was previously mentioned, the initial schedules of the proposed approach were generated using an allocation method before being further optimized. The optimization process was performed twice; hence the name First and Second Order Optimization as can be seen in the table.

The results obtained from the experiments were recorded in Table 5-1. Based on the results, we can state that a significant improvement has been achieved for each dataset from the initial cost to the cost produced by GA (from first to last iteration). In addition, the time taken for the GA optimization was also very efficient, with the average of less than 1.5 second of CPU time for each dataset. However, no single combination of parameters managed to outperform the results gained by HC in our previous study as reported in Chapter 4. These improvements were obtained for all combinations of parameters, therefore, the fact that GA is an effective optimization technique through exams slot swapping, was

proven. The cost was further reduced by reassigning the exams in the first order optimization and was further improved during the second order optimization for some datasets. This pattern can be observed if one moves from left to right across this table.

Table 5-1: Results Obtained Using GA Optimization With Minimization of Total Slots Conflicts and Group Reassignments on Toronto Benchmark Problem

Dataset (Initial Cost)	npar	Max Iter	First Order Optimization				Second Order Optimization			
			Cost at First Iteration (1)	Cost at Last Iteration (10/15/20)	CPU Time (sec)	Cost after Reassign	Cost at First Iteration (1)	Cost at Last Iteration (10/15/20)	CPU Time (sec)	Cost after Reassign
car-f-92 (I) (9.4318)	**10**	10	7.8894	6.7645	0.296	5.2512	5.2512	5.2251	0.873	5.1215
		15	7.8894	6.7589	0.328	5.2512	5.2512	5.2251	0.796	5.1215
		20	7.8894	6.7589	0.484	5.2512	5.2512	5.2251	0.702	5.1215
	15	10	8.0423	6.7761	0.609	5.2512	5.2512	5.2512	1.232	5.2512
		15	8.0423	6.7761	0.921	5.2512	5.2512	5.2512	1.513	5.2512
		20	8.0423	6.7761	1.248	5.2512	5.2512	5.2512	1.622	5.2512
	20	10	8.0016	6.7523	1.123	5.2512	5.2512	5.2438	1.731	5.1197
		15	8.0016	6.7113	1.591	5.2512	5.2512	5.2438	2.403	5.1197
		20	8.0016	6.7113	2.231	5.2512	5.2512	5.2438	2.075	5.1197
car-s-91 (I) (11.7678)	**10**	10	9.3936	8.0188	0.234	6.5652	6.5018	6.3214	1.389	6.3083
		15	9.3936	7.9814	0.405	6.5652	6.5018	6.3214	1.076	6.3083
		20	9.3936	7.9814	0.546	6.5652	6.5018	6.3214	1.513	6.3083
	15	10	9.074	7.7664	0.593	6.5652	6.5018	6.3046	2.059	6.1502
		15	9.074	7.7563	0.858	6.5652	6.5018	6.3046	1.482	6.1502

		20	9.074	7.7563	1.295	6.5652	6.5018	6.3046	1.529	6.1502
	20	10	9.0604	7.6413	1.139	6.5652	6.5018	6.2936	1.482	6.2348
		15	9.0604	7.6413	1.934	6.5652	6.5018	6.2936	2.543	6.2348
		20	9.0604	7.6413	2.574	6.5652	6.5018	6.2936	3.791	6.2348
ear-f-83 (I) (72.6889)	**10**	10	56.4729	49.1911	0.171	47.0667	45.0222	44.4204	0.203	43.7547
		15	56.4729	49.1911	0.301	47.0667	45.0222	44.4204	0.311	43.7547
		20	56.4729	49.1911	0.39	47.0667	45.0222	44.4204	0.421	43.7547
	15	10	53.5138	48.9929	0.499	47.0667	45.0222	43.7209	0.499	43.5369
		15	53.5138	48.9929	0.677	47.0667	45.0222	43.7209	0.72	43.5369
		20	53.5138	48.9929	0.998	47.0667	45.0222	43.7209	1.029	43.5369
	20	10	53.712	49.3271	1.014	47.0667	45.0222	42.7778	0.874	40.9298
		15	53.712	49.3271	1.498	47.0667	45.0222	42.7778	1.342	40.9298
		20	53.712	49.3271	1.81	47.0667	45.0222	42.7778	2.028	40.9298
hec-s-92 (I) (21.8771)	**10**	10	19.3213	15.1902	0.156	15.011	15.011	13.4853	0.172	13.3808
		15	19.3213	15.1902	0.281	15.011	15.011	13.4853	0.28	13.3808
		20	19.3213	15.1902	0.359	15.011	15.011	13.4853	0.343	13.3808
	15	10	19.3932	14.1367	0.453	15.011	14.6394	12.9954	0.421	12.8696
		15	19.3932	14.1367	0.593	15.011	14.6394	12.9954	0.639	12.8696
		20	19.3932	14.1367	2.817	15.011	14.6394	12.9954	2.601	12.8696
		10	18.9968	13.622	15.147	15.011	14.3943	12.91	11.747	12.8232

	20	15	18.9968	13.622	1.108	15.011	14.3943	12.91	1.264	12.8232
		20	18.9968	13.622	1.56	15.011	14.3943	12.91	1.497	12.8232
kfu-s-93 (37.7923)	**10**	10	25.8813	18.5782	0.188	17.5764	17.5764	17.5764	0.453	17.5556
		15	25.8813	18.5782	0.297	17.5764	17.5764	17.5764	0.593	17.5556
		20	25.8813	18.5782	0.375	17.5764	17.5764	17.5764	0.515	17.5556
	15	10	25.2875	19.0258	0.468	17.5764	17.5764	17.5764	0.686	17.5556
		15	25.2875	19.0258	0.687	17.5764	17.5764	17.5764	0.967	17.5556
		20	25.2875	19.0258	0.921	17.5764	17.5764	17.5764	1.373	17.5556
	20	10	25.2875	18.1512	11.341	17.5764	17.5764	17.5014	15.632	17.1342
		15	25.2875	18.1512	1.326	17.5764	17.5764	17.3632	1.498	16.9226
		20	25.2875	18.1512	1.716	17.5764	17.5764	17.3632	1.84	16.9226
lse-f-91 (23.7689)	**10**	10	20.0411	17.0422	0.172	16.8375	16.8375	16.0503	0.343	15.814
		15	20.0411	17.0422	0.28	16.8375	16.8375	16.0503	0.437	15.814
		20	20.0411	17.0422	0.343	16.8375	16.8375	16.0503	0.515	15.814
	15	10	19.4017	17.1988	0.437	16.8375	16.8375	15.7095	0.608	15.6669
		15	19.4017	17.1988	0.655	16.8375	16.8375	15.7095	0.936	15.6669
		20	19.4017	17.1988	0.765	16.8375	16.8375	15.7095	1.029	15.6669
	20	10	19.099	16.135	0.92	16.8375	16.8375	15.6552	0.811	15.1555
		15	19.099	16.135	1.185	16.8375	16.8375	15.6552	1.248	15.1555
		20	19.099	16.135	1.684	16.8375	16.8375	15.6552	1.747	15.1555

rye-f-92 (31.4992)	**10**	10	20.4928	18.1311	0.203	11.2954	11.2954	11.1047	0.483	10.8862
		15	20.4928	18.1311	0.312	11.2954	11.2954	11.1047	0.561	10.8862
		20	20.4928	18.1311	0.39	11.2954	11.2954	11.1047	0.764	10.8862
	15	10	19.0379	16.4612	0.509	11.2954	11.2954	10.9618	1.011	10.9263
		15	19.0379	16.4612	0.733	11.2954	11.2954	10.9618	1.139	10.9263
		20	19.0379	16.4612	0.983	11.2954	11.2954	10.9618	1.388	10.9263
	20	10	18.5681	17.3608	0.983	11.2954	11.2954	10.9618	1.373	10.9263
		15	18.5681	17.3608	1.358	11.2954	11.2954	10.9618	1.857	10.9263
		20	18.5681	17.3608	1.763	11.2954	11.2954	10.9618	2.325	10.9263
sta-f-83 (I) (201.0638)	**10**	10	175.8216	163.9869	0.156	169.9624	169.707	161.7021	0.156	159.9591
		15	175.8216	163.9869	0.249	169.9624	169.707	161.7021	0.265	159.9591
		20	175.8216	163.9869	0.328	169.9624	169.707	161.7021	0.328	159.9591
	15	10	172.8052	163.1211	0.39	169.9624	169.4779	161.198	0.39	161.1309
		15	172.8052	163.1211	0.561	169.9624	169.4779	161.198	0.578	161.1309
		20	172.8052	163.1211	0.749	169.9624	169.4779	161.198	0.748	161.1309
	20	10	170.4517	161.8756	0.733	169.9624	164.6547	160.9264	0.655	160.4583
		15	170.4517	161.8756	0.983	169.9624	164.6547	160.9264	1.076	160.4583
		20	170.4517	161.8756	1.311	169.9624	164.6547	160.9264	1.389	160.4583
tre-s-92 (14.811)	**10**	10	12.6252	11.4214	0.188	10.8487	10.8487	10.6279	0.234	10.3505
		15	12.6252	11.4214	0.312	10.8487	10.8487	10.6279	0.343	10.3505

		20	12.6252	11.4214	0.406	10.8487	10.8487	10.6279	0.39	10.3505
	15	10	12.5101	11.0069	0.452	10.8487	10.8487	10.575	0.484	10.495
		15	12.5101	11.0069	0.764	10.8487	10.8487	10.575	0.686	10.495
		20	12.5101	11.0069	0.968	10.8487	10.8487	10.575	0.873	10.495
	20	10	12.2224	10.9574	0.936	10.8487	10.8487	10.5089	0.858	10.3524
		15	12.2224	10.9574	1.248	10.8487	10.8487	10.5089	1.17	10.3524
		20	12.2224	10.9574	1.841	10.8487	10.8487	10.5089	1.545	10.3524
uta-s-92 (I) (7.7053	**10**	10	5.9236	5.1598	0.249	4.2319	4.2319	4.2155	1.139	4.023
		15	5.9236	5.1337	0.422	4.2319	4.2319	4.2155	1.201	4.023
		20	5.9236	5.1337	6.693	4.2319	4.2319	4.2155	15.88	4.023
	15	10	5.9267	5.0116	0.562	4.2319	4.2319	4.1421	1.216	4.0687
		15	5.9267	5.0116	0.842	4.2319	4.2319	4.1365	1.155	3.9579
		20	5.9267	5.0116	1.029	4.2319	4.2319	4.1365	1.638	3.9579
	20	10	5.9	5.0359	1.17	4.2319	4.2319	4.1563	1.529	4.0837
		15	5.9	5.0356	1.56	4.2319	4.2319	4.1563	2.277	4.0837
		20	5.9	5.0356	1.903	4.2319	4.2319	4.1563	1.981	4.0837
ute-s-92 (56.9698)	**10**	10	38.388	34.5404	0.124	31.5378	31.5378	31.5378	0.125	31.5378
		15	38.388	34.5404	0.219	31.5378	31.5378	31.5378	0.281	31.5378
		20	38.388	34.5404	0.281	31.5378	31.5378	31.5378	0.265	31.5378
		10	35.7745	32.956	0.327	31.5378	31.5378	31.4724	0.312	29.3473

	15	15	35.7745	32.956	0.499	31.5378	31.5378	31.4724	0.453	29.3473
		20	35.7745	32.956	0.64	31.5378	31.5378	31.4724	0.624	29.3473
	20	10	35.7745	32.8549	0.624	31.5378	31.5378	31.4724	0.639	29.3473
		15	35.7745	32.8549	0.843	31.5378	31.5378	31.4724	0.92	29.3473
		20	35.7745	32.8549	1.294	31.5378	31.5378	31.4724	1.186	29.3473
yor-f-83 (I) (59.0404)	**10**	10	51.6706	47.543	0.156	44.8151	44.8151	43.0085	0.203	42.6291
		15	51.6706	47.543	3.682	44.8151	44.8151	43.0085	3.619	42.6291
		20	51.6706	47.543	0.343	44.8151	44.8151	43.0085	0.359	42.6291
	15	10	50.7736	47.5016	0.437	44.8151	43.3199	42.831	0.405	42.5409
		15	50.7736	47.5016	0.593	44.8151	43.3199	42.831	0.592	42.5409
		20	50.7736	47.5016	0.78	44.8151	43.3199	42.831	0.764	42.5409
	20	10	51.9692	46.6227	0.795	44.8151	44.3496	42.831	0.764	42.5409
		15	51.9692	46.6227	1.155	44.8151	44.3496	42.831	1.107	42.5409
		20	51.9692	46.6227	1.466	44.8151	44.3496	42.831	1.513	42.5409

In the results presented, if an observation of the cost produced by the same number of parents at Iteration 1 is made, one can clearly see that the costs that were produced for all cases were the same, for example for car-f-92 (I), where the values for the first 3 rows in the table are 7.8894.

A general assumption that can be made is that, for the same number of parents, an increase in the number of iterations does not offer an advantage in terms of improving the quality of the schedules (or reducing the cost) because, for most of the datasets, the costs remain even though the number of iterations were increased. There are some exceptions to this case however, i.e. for car-f-92 (I), car-s-91 (I), and uta-s-92 (I).

With the increased factor, in terms of the increase in the number of parents, it can also be deduced that the costs produced are mostly reductions or, in other words, improvements for the majority of the datasets, except for car-f-92 (I), ear-f-83 (I), kfu-s-93, lse-f-91, rye-f-92, and uta-s-92 (I) with npar=15, 20, 15, 15, 20 and 15 respectively. This can be seen in Figure 5.4 and Figure 5.5 which illustrate Carter Cost (2.1) vs. Number of Parents for datasets sta-f-83 and ute-s-92.

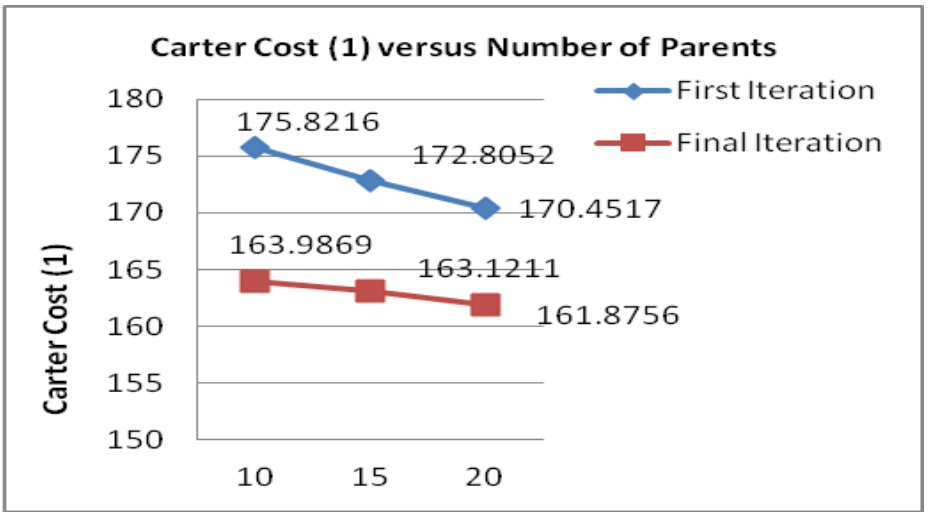


Figure 5.4: Carter Cost (2.1) vs. Number of Parents for sta-f-83 dataset

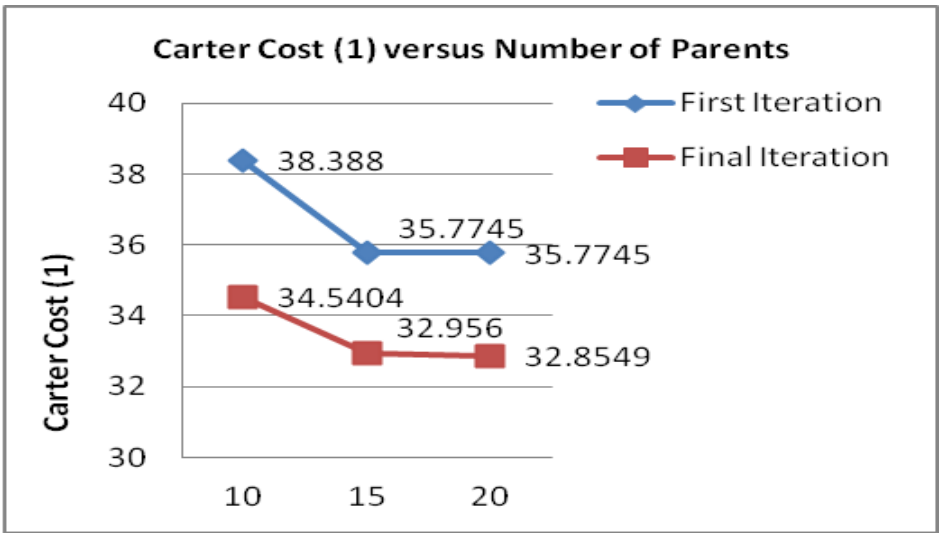


Figure 5.5: Carter Cost (2.1) vs. Number of Parents for ute-s-92 dataset.

For the second order of optimization, the behaviour of the results produced by GA is almost the same as the first order optimization, both when moving horizontally (left to right) and vertically (top to bottom) across the table. Most importantly, for all the datasets, the costs have

been significantly reduced before performing GA optimization and from the first iteration to the last iteration.

Based on the results recorded, it can be concluded that the values selected for both parameters (i.e. the number of parents and iterations) in the GA proposed for the slot reordering of the exam schedules are suitable and the quality of the schedules was improved. In most cases, the values of both parameters were increased and that this assisted the procedure in exploring the search space efficiently and escaping from its local optima.

Moreover, it has been observed that, for npar = 10 the cost produced by the GA is less encouraging than using npar = 15 or 20. Even though the GA was run over a few iterations with 10 parents, it appears that the explorations of the search space only managed to find its local optima. Therefore, to prevent the GA from getting stuck in its local optima, according to the overall results, it is suggested to use npar = 15 to 20 with the same range of iterations. These suggested values seem to generate better results and have a better chance of arriving at their global minima.

Based on the above results and observations, the results obtained by using Hill Climbing and the Genetic Algorithm optimization on the initial feasible schedule generated by the allocation method before performing other optimization are shown in Table 5-2. For the Hill Climbing, the worst and the best costs during the process were recorded (permutations of slots), and for the Genetic Algorithm, the cost produced after Generation 1 (Gen 1) and Generation 15 (Gen 15) is presented.

The best cost produced for each type of optimization is accepted and the ordering of the slots was rearranged accordingly before performing further optimization: i.e. reassignment of exams between slots and repeating the whole set of the optimization process later until no further improvement in the schedule cost is evident. The accepted cost together with the CPU time taken for each process can be seen in these tables.

Based on the results presented in Table 5-2, it can be seen that the proposed Greedy Hill Climbing (HC) method outperformed the GA in all cases during the optimization when tested on the benchmark datasets; all the results produced by the GA for all the datasets after generation 15 (Gen 15) were outperformed by the results produced by HC.

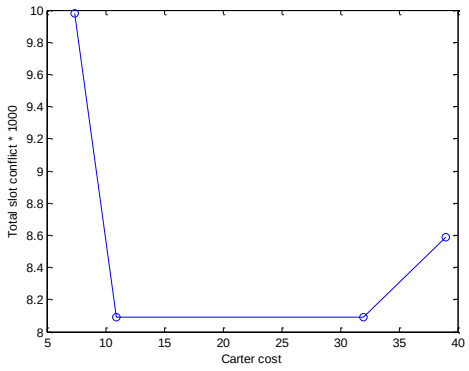
It should be highlighted that the cost obtained by the GA for all datasets at generation 1 (Gen 1) are quite encouraging and are much lower than the worse cost that was obtained by HC. However, all of them failed to improve on the cost obtained by HC after generation 15 (Gen 15).

Using the data that was gathered from the experiments on all the datasets, graphs for the cost (2.1) versus the Total Slot Conflicts for all benchmark datasets were plotted as shown in Figure 5.6. Diagram (a1), (b1), (c1), (d1), ….. (n1) are the graphs (continuous line-graphs) when HC optimization was used whereas diagrams (a2), (b2), (c2), (d2),…… (n2) are the graphs (dashed line-graphs) plotted when the GA optimization was used. The diagrams in these figures are arranged according to the sequence of datasets in Table 5-2.

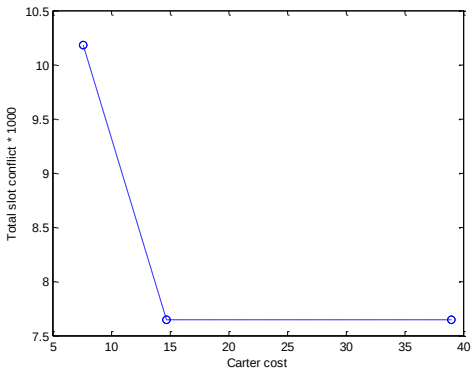
Table 5-2: Comparison of Results Obtained By Using Hill Climbing and Genetic Algorithm Optimization on Nott and Toronto Datasets

Dataset / Initial Cost	**Hill Climbing**				**Genetic Algorithm**			
	Worst cost	**Best cost**	**Accepted Cost Before Further Optimization**	**CPU Time (seconds)**	**Gen 1**	**Gen 15**	**Accepted Cost Before Further Optimization**	**CPU Time (seconds)**
Nott / 38.99	31.95	10.94	**10.94**	187.27	28.03	14.74	14.74	3.39
car-f-92 / 9.43	8.89	5.36	**5.36**	268.97	8.07	6.68	6.68	5.11
car-s-91 / 11.77	10.43	6.26	**6.26**	351.39	9.37	8.10	8.10	5.99
ear-f-83 (I) / 72.69	62.57	40.45	**40.45**	136.77	53.51	48.99	48.99	1.78
hec-s-92 (I) / 22.83	22.55	12.52	**12.52**	27.52	19.39	14.14	14.14	2.30
kfu-s-93 / 37.79	29.89	16.06	**16.06**	40.36	26.81	20.06	20.06	2.48

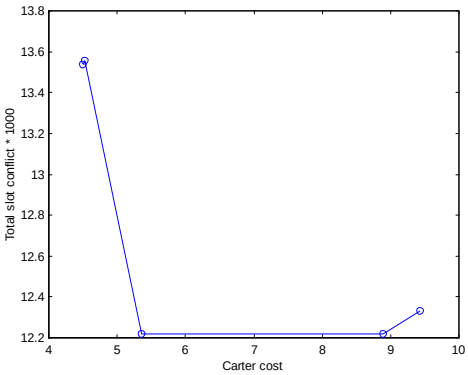
lse-f-91 / 23.77	22.42	14.63	**14.63**	26.59	19.40	17.20	17.20	2.25
pur-s-93 (I) / 14.91	14.27	6.69	**6.69**	321.05	11.94	8.47	8.47	7.97
rye-f-92 / 31.50	28.55	12.68	**12.68**	73.17	19.04	16.46	16.46	3.25
sta-f-83 / 201.95	193.47	158.43	**158.43**	10.28	172.80	163.12	163.12	0.52
tre-s-92 / 14.81	13.25	9.84	**9.84**	66.34	12.76	11.70	11.70	3.17
uta-s-92 (I) / 7.30	6.59	4.23	**4.23**	455.94	6.19	5.22	5.22	6.54
ute-s-92 / 56.97	43.25	31.79	**31.79**	2.91	35.77	32.96	32.96	1.30
yor-f-83 (I) / 59.04	56.31	43.36	**43.36**	46.99	50.77	47.50	47.50	0.75



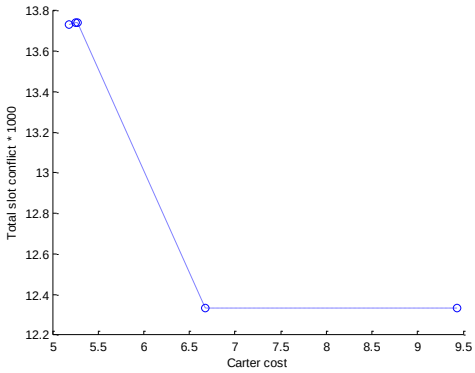
(a1) Nott



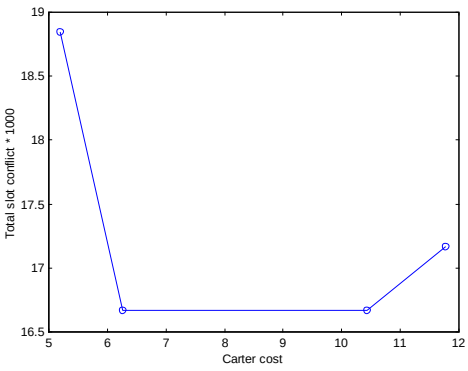
(a2) Nott



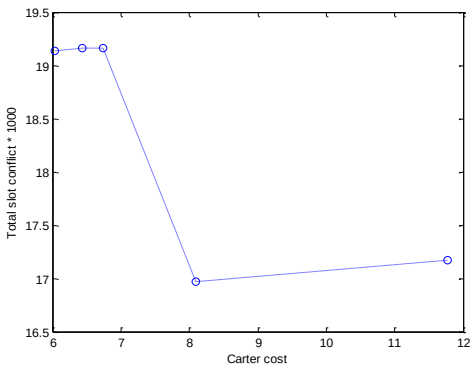
(b1) car-f-92



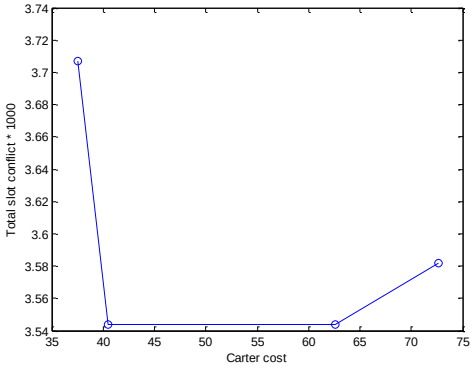
(b2) car-f-92



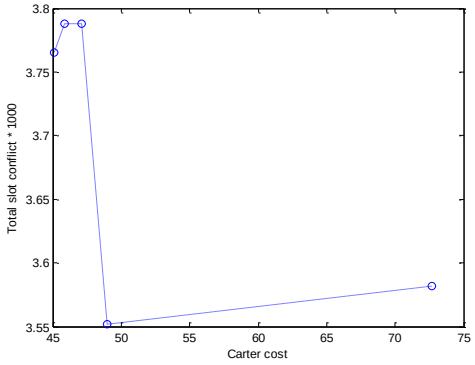
(c1) car-s-91



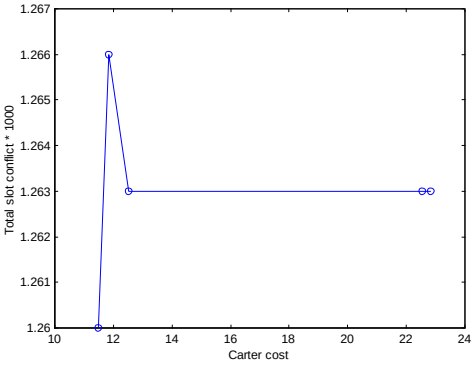
(c2) car-s-91



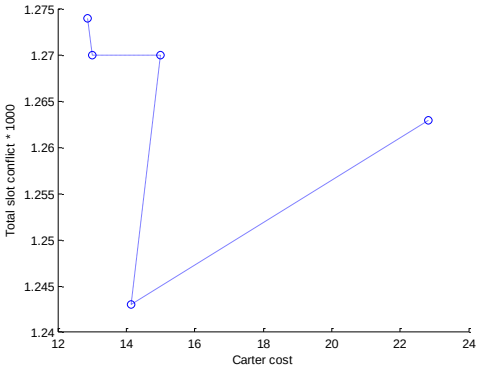
(d1) ear-f-83 (I)



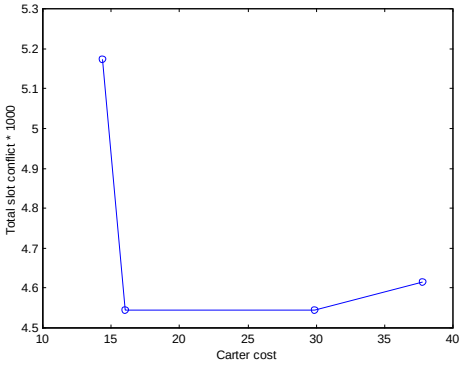
(d2) ear-f-83 (I)



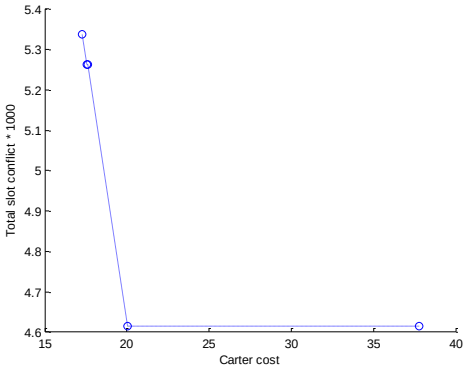
(e1) hec-s-92 (I)



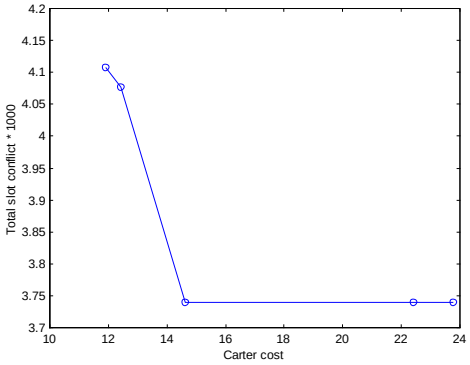
(e2) hec-s-92 (I)



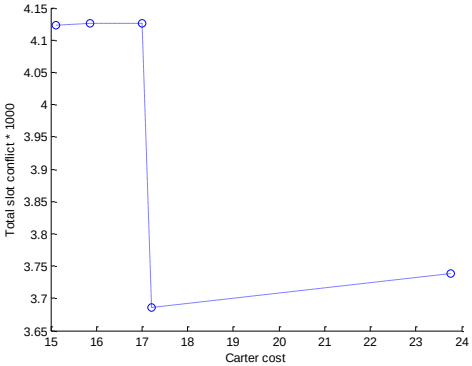
(f1) kfu-s-93



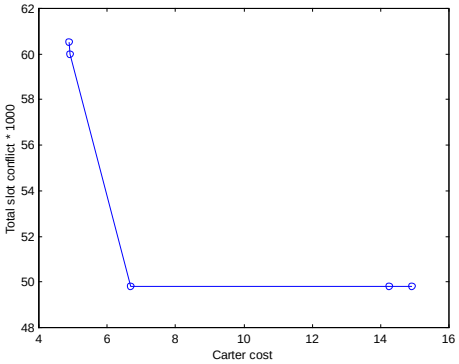
(f2) kfu-s-93



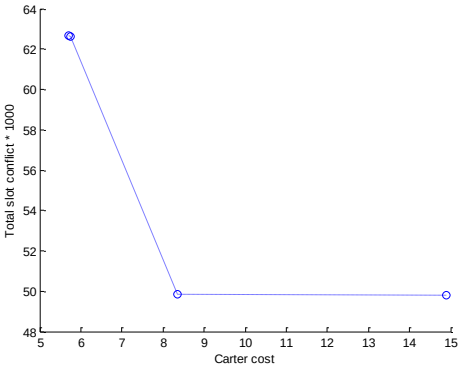
(g1) lse-f-91



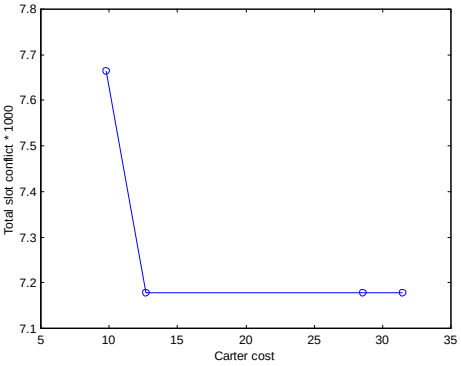
(g2) lse-f-91



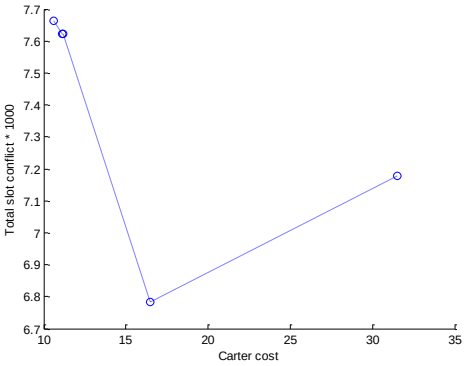
(h1) pur-s-93 (I)



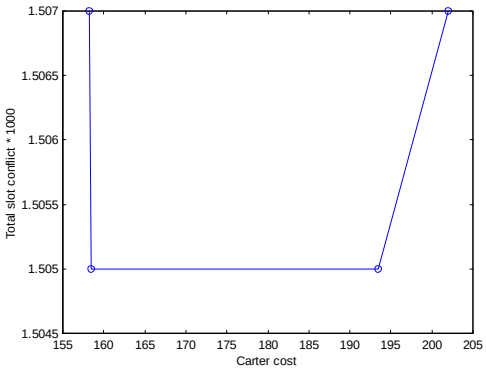
(h2) pur-s-93 (I)



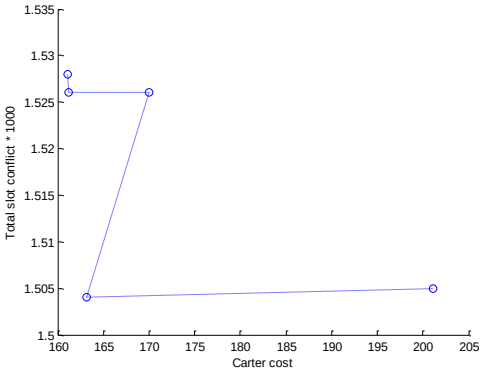
(i1) rye-f-92



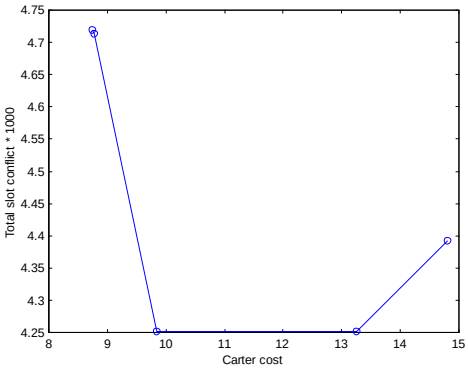
(i2) rye-f-92



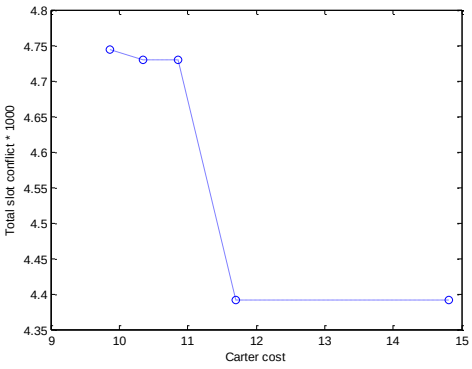
(j1) sta-f-83



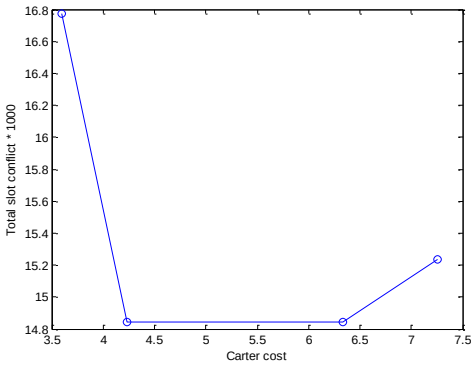
(j2) sta-f-83



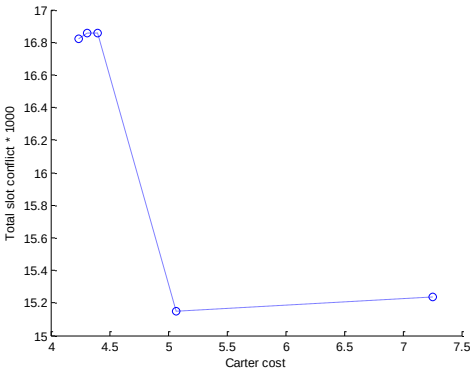
(k1) tre-s-92



(k2) tre-s-92



(l1) uta-s-92 (I)



(l2) uta-s-92 (I)

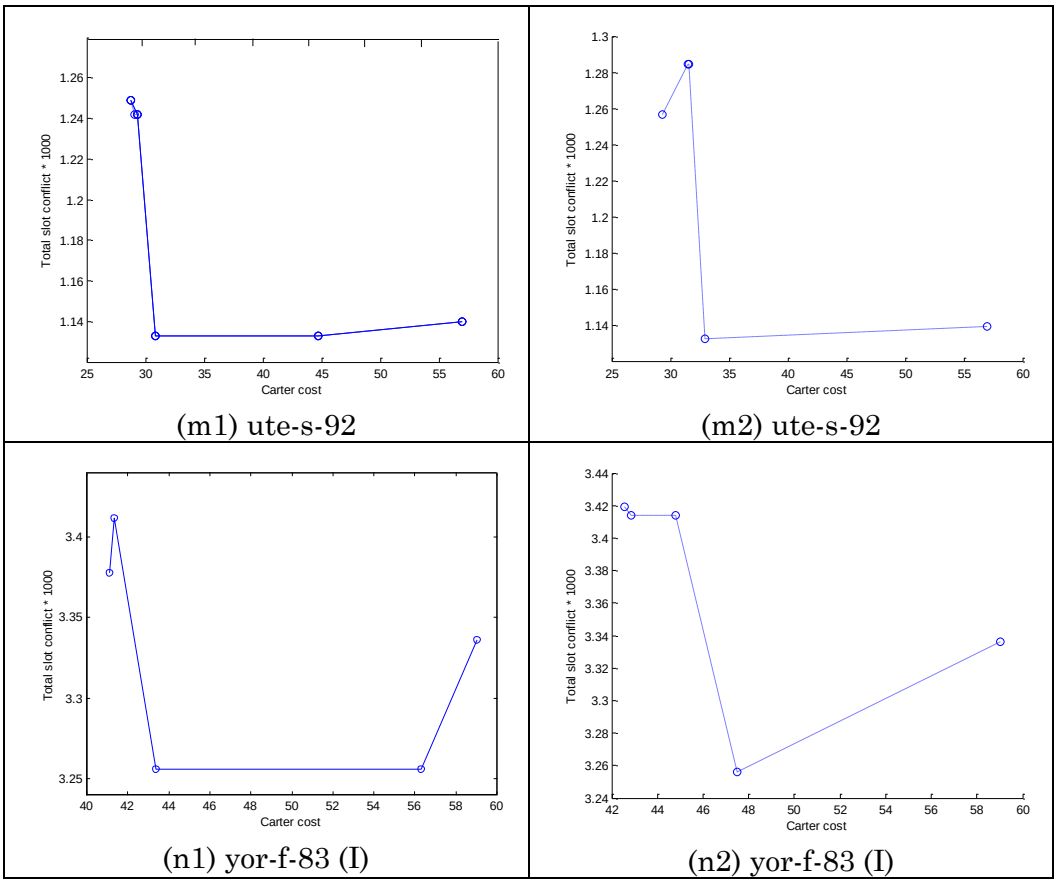


Figure 5.6: Cost (2.1) vs. the Total Slot Conflicts for Benchmark Datasets (Using Hill Climbing (HC) vs. Genetic Algorithm (GA)). Note: Continuous line- graphs on the left of this figure are for HC and dashed line –graphs are for GA.

From the graphs presented above, it can be noted that the pattern obtained from both HC and GA optimizations are similar and deterministic. The horizontal line constructed from the second data point to the third data point in the diagram (a1) to (n1) is due to a reduction of cost via the permutations of exam slots (Greedy HC) which did not involve any augmentation of total slot conflicts. The dotted line from the first data point to the second data point in each diagram (a2) to (n2) is constructed based on the GA optimization discussed earlier in this paper.

A significant reduction in terms of the initial cost has been achieved by performing the GA optimization as shown in the dotted lines at this stage. These lines also showed that the proposed GA managed to substitute the HC implementation and was incorporated successfully in the whole set of our optimization process.

One of the things that can be seen in the graphs is that the line constructed by GA optimization is not always horizontal indicating a change in the value of the total slot conflicts. This is because the crossover and mutation of exam slots in the GA optimization process changed the assignment of some exams to slots to ensure the feasibility of the schedules. In changing the exam from one slot to another, it changes the value of total slot conflicts due to the interactions of the shifted exam with other exams that have common students. This is not the case during HC optimization where the total exam-slot conflicts does not change because the individual exams to slots remain as they were before the permutations.

Based on our observations, GA produced quite encouraging results, however it converges quite quickly, whereby in most datasets, (out of the 9 combinations) best Carter cost (2.1) was obtained as early as when using 10 parents and 15 iterations. Even though more parents and iterations were used, however they did not record any improvements. This is mainly because the GA procedure was being used in the permutations of exams slots, where the number of slots are normally quite small (ranging from 10 to 42 only). Thus the search space for the permutations is small.

In the experiments conducted, overall we have obtained very good results using GA, which demonstrates a deterministic optimization pattern of the proposed framework. From initial feasible solution constructed at the earlier stage, after going through the GA procedures, a very significant reduction in terms of Carter cost (2.1) was obtained successfully as predicted. The reduction can be seen obviously in the horizontal line in each graph presented in the thesis. This shows that the idea of optimizing the schedules using permutations of exams slots is very efficient and reliable.

Based on the results, also, it indirectly shows that the framework, which consists of a few stages (i.e. pre-processing, scheduling and optimization), is proven to be an effective and flexible framework where some procedures can be replaced and incorporated into the framework adeptly. This was shown by the successful substitution of the GA with the existing HC.

A point that should be noted is the computational time taken to execute both methods. Even though the GA did not surpass HC in all cases the time taken to execute the process was remarkably short when compared to the implementation of HC. The proposed GA was simple, straightforward, and quite effective and took a short amount of time to improve the initial feasible schedule, although a majority of the research claims that GA takes a very long time to solve scheduling problems (for example, as claimed by (Abramson, 1992)).

This advantage (the minimal time requirement) could offer even more advantages, because, based on the results presented earlier, it can be predicted that an addition to the number of iterations or generations (with the aim to generate better offsprings) to the GA execution will only add a short amount of computational time, which can be considered to be acceptable.

Unfortunately however this is not the case. No benefits, in terms of reductions to the cost of the exam schedule generated, were given by increasing the number of generations. Experiments were conducted up to 20 generations, but the highest number of generations at which the cost is reduced is generation 12 (car-f-92(I)). A second round of optimization was performed in order to test whether it could reduce the cost further. Therefore, after performing reassignment of exams on the schedules obtained by the GA optimization, we repeated the GA optimization one more time. As can be seen in Table 5-3, the highest number of generations that could reduce the schedule cost is generation 11 (rye-f-92) even though 15 generations were tested.

The cost (2.1) was improved upon in the second round for most of the datasets (except for nott, carf92, kfus93, purs93, and utes9) and this is illustrated in diagram (a2) to (n2) by the third data point to the fourth data point.

The final results obtained by HC and GA methods, which were further improved by reassignments of exams, were compared and can be found in Table 5-4. It can be clearly seen that HC outperforms GA in all cases, even though the execution time recorded was somewhat high in comparison to the GA, but the amount of time taken was still reasonable which is only a few hundreds seconds of CPU time.

Table 5-3: Number of Generations That Improved the Schedule Cost During GA

Dataset	First Order Optimization: Cost Improved Until Iteration	Second Order Optimization: Cost Improved Until Iteration
notts	7	0
carf92	12	0
cars91	9	2
earf83	7	6
hecs92	6	7
kfus93	10	0
lsef91	8	4
purs93	11	0
ryef92	8	11
staf83	6	4
tres92	8	5
utas92	10	4
utes92	3	0
yorf83	6	3

Table 5-4: Final Cost Produced Using HC versus GA Optimization

Dataset	Final Cost Produced after All Optimization Processes For HC (Rahim *et al.*, 2012)	Final Cost Produced after All Optimization Processes For GA (Rahim *et al.,* 2013a)
notts	7.34	7.62
carf92	4.49	5.18
cars91	5.19	6.03
earf83	37.57	45.08
hecs92	11.47	12.90
kfus93	14.36	17.27
lsef91	11.90	15.11
purs93	4.88	5.57
ryef92	9.8	10.63
staf83	158.25	161.13
tres92	8.74	9.86
utas92	3.58	4.01
utes92	27.37	29.35
yorf83	41.10	43.52

To summarize, the GA has been successfully implemented and incorporated into the proposed Domain Transformation Approach framework to solve the examination scheduling problems. The GA proposed in this study is an efficient algorithm that managed to achieve its objective which is to improve the initial feasible exam schedules (before being optimized). With a robust implementation, it managed to

explore the search space efficiently and produced a good quality timetable with a fast execution time. Since it succeeded in improving the quality of the initial feasible schedule in a very efficient time and it produced very consistent results with a deterministic optimization pattern, we consider the incorporation of GA into our proposed framework is very effective. However, this procedure works best within a certain range of parameters (and depends on careful parameter tweaking). For this particular GA (for slot swapping), it is suggested that at least 15 to 20 parents and also 15 to 20 iterations be used in order to get the best solutions.

Furthermore, the cost that was arrived at through GA optimisation did not improve on the results that were obtained with the Greedy HC optimisation. Although the computational time taken by the GA execution is a lot shorter than HC, an additional reasonable amount of time should be taken in order to ensure that good quality schedules are obtained. HC managed to improve the initial feasible schedule without fail for all datasets and always surpassed the GA results, therefore, it is suggested that the proposed HC be used as a basis of the optimization processes.

Through the findings of this research the claim made by Ross *et al.* (1998) that sometimes the GA is not a very good approach to solving problems, is supported by empirical evidence.

Even though the proposed approach has produced very encouraging and consistent results, as reported in the previous chapter, it is still recommended that a possibility of improving the solutions be sought. Once the fact that the Greedy Hill Climbing slots permutations are merely a local search procedure is realized then it is considered desirable that a global search procedure be incorporated in the proposed framework. For this purpose, the Genetic Algorithm (GA) was chosen to reorder the slots in the spread matrix (with the aim of maximizing the gap between consecutive exams). Even though GA is highly dependent on the adjustment of parameters, the fact that it was reported as capable of producing high quality examination schedules, provided a motivation for our investigation of GA in the context of the proposed examination scheduling. As such, GA was implemented and has substituted for the traditional Greedy Hill Climbing. From the experiments conducted on benchmark datasets, it was revealed that Greedy Hill Climbing outperformed the GA in all cases, where the costs obtained by HC were considerably lower in all problems. Though the technique of slot permutations proposed using HC is just a local search procedure it was concluded that our approach of restarting the search from different starting points managed to explore the search space efficiently.

CHAPTER 6

Conclusions and Future Work

This chapter presents the summary of the work conducted in this thesis. The proposed approach to solving university examination timetabling problem is summarized in brief. The contributions made to the area of examination timetabling are also highlighted. Consequently, future research directions based on the proposed approach are also discussed.

6.1 Summary of the Research

In this thesis, we presented and discussed our proposed approach to solving the university's examinations scheduling problems. In our initial study, we believe that when enough information is supplied, using a systematic method, the real world examination scheduling problems can be solved efficiently and effectively. Even though the real world university examination scheduling problems are complex and very challenging to solve, we are certain that by imitating how people solve complex problem solutions are always reproducible. When people deal with problems, first they look at them in a macro perspective or bigger picture in order to grasp the essence of the problem, before focusing on the minute details. When deriving this simplified "big picture", people

will normally do some information pre-processing in order to group related data and by doing so to avoid being drawn into detail. Such a simplification can take various forms but the common characteristic is a deliberate tradeoff between the accuracy and generality of problem representation. The most easy and least strenuous way of problem simplification is when the problems is divided and then solved in stages.

Inspired by human approach to solving problems, we have proposed a *Domain Transformation Approach* in solving the university examination scheduling problems. The *Domain Transformation Approach* that was proposed, is an approach that transforms the original timetabling problem domain into smaller sub-domains that manages to reduce the problems' complexities effectively. By subdividing the real world examination scheduling problems into smaller sub-problems, each problem was solved more effectively.

We have chosen to solve the examinations scheduling problems by performing a few independent stages in sequence: 1) pre-processing, 2) scheduling, and 3) optimization. This approach is quite similar to *construction and improvement* (Hertz, 1991) but in our approach, it is a bit different, because we introduced a pre-processing step prior to both the scheduling (*construction*) and optimization (*improvement*) phases.

In the early stages of this study, we postulated that pre-processing of data and constraints provides more meaningful information that could be utilized at a later stage in the scheduling process. When performing pre-processing, certain data and constraints would be grouped according to certain criteria (for example: the grouping of conflicting exams and

non-conflicting exams). This will reduce cross-checking and cross referencing in voluminous data during scheduling.

Based on our preliminary study, we learned that information processing was a key element to Granular Computing (Pedrycz *et al.*, 2000; Bargiela and Pedrycz, 2002; Bargiela *et al.*, 2004; Bargiela and Pedrycz, 2008). (The information processing approach is a multilevel processing approach, which is capable of producing a new representation of meaningful data. The basic details of the original data is hidden, thus directly reducing the complexities of the problems.

Inspired by this concept, which we hypothesize could ease the scheduling process significantly, we have taken an approach which consists of an information pre-processing stage, followed by real scheduling and separate optimization steps. The work based on our proposed framework are summarized below:

- ***Pre-processing:*** The very first step taken in our proposed examination scheduling algorithm was the pre-processing of data and constraints prior to the generation of the feasible timetable. This was performed through the abstraction of essential features from the original students/exams data. By performing this step, we successfully mapped the original problem expressed in multi-dimensional space of exams and students onto a reduced dimensionality space. Data was grouped together according to certain criteria and therefore the pre-processed data, for example the *exam conflict matrix* and *spread matrix* were rendered more meaningful. These aggregated data, which could be referred to as *information granules* according to the concept

of Granular Computing (Pedrycz *et al.*, 2000; Bargiela and Pedrycz, 2002; Bargiela *et al.*, 2004; Bargiela and Pedrycz, 2008), supplied us with important information for efficient scheduling. While more time could be saved because cross referencing and checking of data/constraints can be eliminated, the schedule is feasible (in terms of ensuring no conflicting exams are scheduled concurrently) and is generated in a short amount of time. This is guaranteed because the scheduling at a later stage (after pre-processing) will always ensure that the exams that will be assigned to a particular slot are from the same group (group of non-conflicting exams – this is, supplied through pre-processing).

- ***Scheduling:*** After pre-processing, scheduling was done by using a standard Graph Coloring method with Largest Degree ordering. In our proposed allocation method, exams with the highest conflicts are placed first in the first available timeslot and is later moved to other exams with lower conflicts. The allocation of exams was determined by decisions made on the basis of four preferences: assigning conflicting exams to none empty slots, assigning conflicting exams to empty slots, assigning none conflicting exams to none empty slots and assigning none conflicting exams to empty slots. Each of these has the following value 0.4, 0.3, 0.2, and 0.1 respectively. The higher the value, the higher the preference for allocation. A verification procedure was used to verify whether the schedule generated was feasible. Once verified, a splitting and merging process was performed on the schedule. By splitting a slot p and reassigning constituent exams, the total number of slots could be reduced if every exam in slot

p can be allocated to some other slot, i.e. not in conflict with exams in other slots.

- ***Backtracking:*** Backtracking was implemented in this study with the aim of reducing the number of slots after the construction of a feasible but not optimal solution (or infeasible – for example in this research the solution for yorf83 did not fulfil the minimum number of timeslots) in the scheduling stage. The implementation of the backtracking was based on the backtracking's algorithm proposed by Carter *et al.* (1996) but with some modifications. In general, Carter *et al.* (1996) implemented backtracking during the initial placement of exams in cases where exam(s) exist that cannot be scheduled to any of the available slots. The assignment of exams that are in conflict with the unscheduled exams will be undone in order to schedule it. As opposed to our approach, the placement of all exams to their allocated slots has already been completed therefore we attempted to convert an infeasible schedule into a feasible one, for the purpose of reducing 1 slot. A reduction in the number of slots at an early stage is desirable, since the cost can then be minimized at a later stage. An initial schedule with a few slots (less than the requirement) can always be altered into one which fulfils this constraint. We postulate that it could provide a useful buffering space during the optimization involving permutations of slots.

- ***Optimization:*** Rather than attempting direct optimization of assignments of exams to specific time-slots, optimizations are

performed on the feasible (but not optimal) schedules obtained in the previous stage. To minimize the cost, we perform the minimization of total slot conflicts, followed by further optimization on the initial schedule by: the permutations of exam slots and the reassignment of exams between slots.

- ***Minimization of Total Slot Conflicts:*** The notion of a slot conflict is a generalization about the notion of an exam conflict, because in any feasible schedule, conflicting exams will always be assigned to different timeslots. The conflict between exams is a binary property (which can be determined via the domain transformation approach) that remains no matter how many students are enrolled in those exams. We have also considered determining the exam-slot conflict count by summing the number of slots that contain conflicting exams for a particular exam *i*. The exam-slot conflict is a binary property that does not increase in value if exam *i* has several conflicting exams in one slot. The information of the total slot conflicts acts as a measure of the ability to reschedule exams between slots. A high total count means fewer slots are available for scheduling and vice versa. Total slot conflicts can simply be minimized by taking each exam from every slot and reassigning it to a new slot that could reduce the total slot conflicts count if there are any. Minimization has the potential to reduce the cost of the exam schedule, however, in the proposed approach, this process is considered to be an augmentation of the potential for following minimization of the cost of the schedule.

- ***Permutations of Exam Slots:*** Taking into account the definition of the Cater cost (2.1) where students taking exams that are t slots apart, where t = {1, 2, 3, 4, 5,6} will give the penalty weight of 16, 8, 4, 2, 1, and 0 respectively, we can devise the task of optimization of the exam spread as a task of re-arranging the time slots such that the smallest sum of elements on the first minor diagonal can be achieved in the rearranged spread matrix (Rahim *et al.*, 2009; Rahim *et al.,* 2012). With this in mind, large numbers in the spread matrix should be moved to minor diagonals that are of the order 6 or more. The permutations in the spread matrix involve the swapping of slots and the repetition of block shifts. In the Greedy Hill Climbing implementation, the provisional swapping of a slot with all other slots is done and the Carter cost (2.1) is evaluated. The swap will be remembered and the matrix will be updated accordingly if the cost is reduced. Realizing that this optimization may lead to local optima, we have adopted a simple measure of restarting the optimization from several initial orderings and picking the best solution from a pre-defined number of runs.

  Prior to the idea of proposing Greedy Hill Climbing, by exploiting the knowledge about the structure of the cost function, we have initially proposed a scheme for renumbering the timeslots in the spread matrix by proposing two methods, namely, Method 1 and Method 2. These methods which later

contributed to the idea of Greedy Hill Climbing slot permutations, are described below.

The first method focused on extracting the smallest element in each row of the spread matrix and places it in the first minor diagonal and later renumbered the relevant time slots. This method works best if the objective it to minimize the number of adjacent exams.

The second method also concerned the smallest element, but unlike the first one, Method 2 extracted the smallest elements in both rows and columns in the spread matrix. These elements were shifted towards the first minor diagonal and gave a more balanced re-numbering that could minimize the sum of higher minor diagonals better.

- ***Reassignments of Exams:*** In the third stage of optimization, we have performed some exams reassignments with the aim to further reduce the Carter cost (2.1). Exams that make a large contribution to the first minor diagonal entries of the reordered spread matrix are reassigned to slots represented by higher minor diagonals (preferably of order 6 or higher).

- ***Substitution of Greedy Hill Climbing Method:*** While proposing a very systematic approach that we hypothesize could reduce the timetable cost significantly using the greedy Hill Climbing slots permutations in the optimization stage, we tried to substitute this

procedure with other procedures. The substitutions were performed with the aim of analyzing whether they could improve the performance in generating quality feasible schedules in comparison to our existing approach.

- We have implemented the Late Acceptance Hill Climbing (LAHC) algorithm which was claimed to be a very powerful strategy by Burke and Bykov (2008; 2012), to substitute the Greedy Hill Climbing in the optimization phase. In our slot permutations using the LAHC implementation, we used a variable to keep the best cost with an array of length L. During the permutations, we took the best cost whenever we found one and updated the spread matrix accordingly to the new orderings. In contrast to the LAHC implementation, a new solution from a single swap was evaluated against an accepted solution from a swap L steps earlier. This solution was added to the list L if it surpassed the existing solution. We implemented the list as a Round Robin list, modifying items at specific location based on the length of L and the number of generations using the modulus of number of generations with L as the index. The way our LAHC was implemented was a bit different as proposed by Burke and Bykov (2008; 2012) to cater for the smaller search space due to the fact that we implemented the LAHC for slots permutations on the spread matrix of a feasible solution with a small number of timeslots.

- Realizing that both the Greedy Hill Climbing and the Late Acceptance Hill Climbing strategies are local search

procedures, we knew that this might cause the solution search to be stuck in the local optimum. Motivated by the fact that a global search procedure is known to guarantee better results in finding the best possible solution (out of all possible solutions in the search space), we implemented a Genetic Algorithm (GA) to substitute the Hill Climbing procedure. In our GA implementation, the original ordering of slots was reordered in order to find the best arrangement with the lowest cost. The original parent was a data structure that contained the initial ordering of slots. New parents were created by taking a portion of the rows (in blocks) and combining them with another portion from other parents. In generating the offsprings, each parent was crossed over with all other parents at a certain random point *R*. Identical offspring were eliminated and replaced with a mutated data structure (where a random slot *t* was interchanged with a random slot *u*). The best offspring with the lowest cost was selected to be the next parent in the next generation. This process continued for a certain number of iterations. The effect of this procedure is that the initial orderings of slots will be shuffled (quite similar to the effect of slots permutations using Hill Climbing) and the cost will be improved.

6.2 Summary of Results

The approach that we proposed to solving the examination scheduling problem has been proven to be very effective in generating feasible exam timetables. When the original domain was transformed into smaller domains the problem was subdivided rendering it less complex and making it easy to solve. In order to analyze the effectiveness of our proposed framework, we have done experiments on benchmark dataset problems. We have tried to solve the examination timetabling benchmark problems, i.e. University of Toronto, University of Nottingham and International Timetabling Competition 2007 (ITC 2007) datasets.

In the early stages, through pre-processing, we have managed to group together important data from the original students-exams data and created new representations of data, for example, the exams conflict matrix and spread matrix which supplied very important information needed for the scheduling. Having the pre-processed data, lengthy search or cross checking of implicit data can be avoided during scheduling.

The allocation method in the scheduling stage that we used which was based on *Graph Colouring* algorithm with Largest Degree pre-ordering, has always created feasible examination schedules which satisfy the hard constraint (no conflicting exams should be assigned concurrently). The procedure that we applied that manages to split and merge possible timeslots has caused the timetables that we created always fulfil the minimum number of required timeslots (for all datasets except for *yorf83* in the Toronto dataset). After the main scheduling stage, the backtracking procedure was performed on all the datasets to further reduce the number

of slot. Backtracking has successfully decreased the number of required time slots in many datasets, and this offered an advantage both in terms of making sure that the specified number of time slots is not exceeded and providing a buffer-space for slots permutations in the subsequent optimization stage.

The optimization phase which started with the minimization of slot conflicts was performed immediately after the initial feasible schedule was obtained in the scheduling (with backtracking) process. The result of minimizing the total slot conflicts was that more slots were prepared that can be used for the rescheduling of exams. To the best of our knowledge, the potential for the rescheduling of exams has not been quantified in the literature so far, despite it being a key factor enabling the improvement of the initial feasible schedule. During the experiments in the course of this thesis, by minimizing the slot conflicts, the cost of the exam schedule was also minimized. This stage can now be considered primarily as the enhancement of the potential for the subsequent minimization of the cost of the schedule.

In the next stage of optimization, permutations of slots in the spread matrix decreased the Carter cost (2.1) of the initial feasible schedule (which was not optimal) significantly for all the datasets that were experimented with, attaining an average of a 50% improvement. This shows that the Greedy Hill Climbing implemented for the slot permutations is an effective procedure that manages to improve the quality of the feasible exams schedule generated earlier.

We have also implemented the Late Acceptance Hill Climbing (LAHC) strategy to substitute the greedy Hill Climbing in our proposed

framework. This strategy was incorporated successfully in the proposed framework and produced quite encouraging results (on average the performance is on par with the Greedy Hill Climbing). However, it has been observed that the parameters used do not give a consistent performance to all the datasets. In some cases, when the parameter size is increased, the cost obtained is better, but in some cases, the increase in the parameters also increased the cost, which meant it reduced the quality of the existing schedule.

Realizing that the proposed Greedy Hill Climbing is a local search procedure, even though overwhelming improvement was obtained, we were concerned that it might merely find the local optimum in the solution space. As such, the Genetic Algorithm optimization (a global search procedure) was implemented in order to substitute the Greedy Hill Climbing algorithm proposed earlier. This is with the aim that it would improve the Carter cost (2.1) obtained by Greedy Hill Climbing.

The GA optimization involved the reordering of timeslots by utilizing crossover and mutation concepts and at the end of the processes, an improved timetable with new timeslot ordering (reduced Carter cost (2.1)) was achieved.

This GA optimization, when tested on all benchmark datasets, managed to improve the cost of the initial feasible schedule obtained from each scheduling stage. However, it was realized that this procedure works best within a certain range of parameters (and depended on careful parameter tweaking). The good cost obtained by the GA did not manage to outperform the results obtained by utilizing the proposed Greedy HC in all cases (on benchmark datasets).

From the above findings, regarding the results based on the three types of slot re-ordering optimizations, we can see that all three have been successfully implemented and incorporated into our proposed framework. This indirectly shows that the framework, which consists of a few stages (i.e. pre-processing, scheduling and optimization), is proven to be an effective and flexible framework where some procedures can be replaced and incorporated into the framework proficiently. This was shown by the substitution of the LAHC and the GA in the approach.

The first stage of optimization has revealed that our greedy Hill Climbing produced expected results. Despite the fact that the results of the Late Acceptance Hill Climbing strategy that were produced were on par (on average) with Greedy Hill Climbing the parameter used in the LAHC did not demonstrate a consistent performance. Due to this, we recommend that traditional Greedy Hill Climbing be utilized in the slot permutation phase.

Similarly to the LAHC, the GA will not be selected to optimize the timetables according to the outcome of the experiments as it was clearly shown that in all the experiments, Greedy Hill Climbing outperformed the GA in all cases of producing good quality schedules. All costs obtained by Greedy HC are considerably lower when compared to the GA.

The findings indicate that even though Greedy Hill Climbing is just a local search procedure by restarting the search from different starting points iteratively, the search managed to obtain good quality solutions (managed to avoid being stuck in local optima).

After performing the slot permutations, the cost was further reduced by reassigning some exams to other slots. This process exploited

the information in the spread matrix where exams that make large contributions to the first minor diagonal entries of the reordered spread matrix are reassigned to slots represented by higher minor diagonals (preferably of the order 6 or higher). Based on the results that were reported, the cost produced by reassigning some exams managed to further reduce the cost without fail.

The whole stage of the optimization was performed twice, and it was observed that in some datasets, the second round was worthwhile and more improvements were recorded. It is worth noting here that feasible solutions with lower total slot conflicts provide a good basis to minimize the cost via a simple reordering of slots and the later reassignment of exams between slots.

Overall the proposed approach in this study produced very high quality exams schedules. The proposed method for examination timetabling utilising Greedy HC optimisation has shown to deliver consistently competitive results for all benchmark datasets (no matter how large or difficult each dataset is). Comparing the average mean deviation from the best known solution for each benchmark dataset, our method shows that the performance are consistent on all types of problem and at average outperforms all other results except by Carter *et al.* (1996) which is the collection of best of all its approaches. The variance of these deviations is smallest for our method when compared to others reported in the literature. Unlike some constructive methods in the literature that demonstrated an uneven performance, where they performed well on some benchmark problems and less well on others. This is a rather undesirable characteristic from the user's perspective, as there is no way of predicting

the quality of the solution that will be obtained using a particular method on a new dataset. Apart from this, it was shown that several of the best results were obtained by the methods that did not report any results for a few datasets.

In conclusion, it is clear that the *Domain Transformation Approach* proposed in this study is very simple and competitive in terms of generating reliably high quality exam schedules. By transforming the original real world scheduling problem into smaller sub-problems and applying appropriate pre-processing, we managed to reduce the complexities of the problem, thus saving a substantial computational effort compared to other methods that require customized post-processing. We would also like to highlight that an important feature of the proposed optimization is that its deterministic pattern in the results generated for all datasets is always preserved, which makes it a novel contribution to the examination scheduling research field.

6.3 Contributions

The following contributions to the field of examination scheduling were presented in this thesis:

- **Reduced complexity of the problem domain.** The *Domain Transformation Approach* which was proposed based on the insights derived from the Granular Computing concept that has clearly transformed the real world examination scheduling problem into smaller problem domains, allowing the problems to be

conquered or solved in stages, making it a less complex problem that can always be solved in a reasonable amount of time.

- **Reduction of problem space.** *Pre-processing* of constraints has grouped together certain data which provided very useful information. The creation of the *exam conflict matrix* and the *spread matrix* from the pre-processing stage has managed to reduce the laborious searching that was required during scheduling. These matrices are very easy to generate and can be used as new data representations in any examination scheduling algorithm.

- **Ensuring feasible solutions**. *Allocation of exams to slots* and *split and merge* procedures successfully created feasible exam schedules (without fail) with encouraging figures in terms of number of slots and cost.

- **Efficiency**. *Backtracking* procedure (Carter *et al.*, 1996) which is an improved algorithm that was proposed and managed to further reduce the number of timeslots of the initial feasible schedule.

- **Optimization procedures**. The *Optimization* stage that consists of three steps: minimization of total slot conflicts, permutation of slots and reassignment of exams were proven to be very effective

procedures at optimizing the initial feasible exam schedules. A significant reduction in costs for all datasets was recorded.

- **Minimization of Total Slot Conflicts approach.** *Minimization of Total Slot Conflicts* has helped to reduce the cost of the exam schedule. This process was considered as an augmentation of the potential for following minimization of the cost of the schedule. Concisely, when the total of slot conflicts is low, on average, there are more slots that can be used for the rescheduling of exams. To the best of our knowledge, the potential for the rescheduling of exams has not been quantified in the literature despite it being a key factor enabling the improvement of the initial feasible schedule.

- **Permutation of slots approach**. *The Permutation of slots*, which was implemented as a variant of the Greedy Hill Climbing algorithm, managed to produce very encouraging results on par with other results documented (based on other constructive methods) in the literature. Even though it was just a local search procedure, our approach of restarting the search from different starting points, managed to outperform the Genetic Algorithm optimization (a global search procedure).

- **Robust scheduling framework.** *The proposed framework* in this study is very systematic, efficient, robust and is proven to be very *flexible*. This was demonstrated by the success of substituting other

procedures in the framework, i.e. the Late Acceptance Hill Climbing and the Genetic Algorithm. Both procedures managed to produce quite good results. This indirectly shows that every stage in our proposed framework is independent and could therefore be integrated with other scheduling approaches in this area.

- **Consistent performance**. *Through the avoidance of exhaustive exploration* of the search space which normally deploys random selection between alternative choices during the optimization process, the approach is capable of generating solutions that are *reproducible* and *consistent*. This feature exhibits that the proposed approach managed to raise the generality of the examination scheduling algorithm, which is universal and applicable to a wide range of university examination scheduling problem.

- **Deterministic optimization pattern.** *Deterministic optimization pattern* obtained for all benchmark datasets is an overwhelming achievement. The approach carried out in the research resulted in a similar pattern for all datasets as described in Section 4.14, 4.15 and 4.16. This behaviour implies a *deterministic optimization pattern* for all datasets resulting to an overwhelming achievement since there are no claims made by other researchers resulting in a deterministic pattern for optimization making the proposed optimization a *novel* contribution to this field.

6.4 Future Work

It would be very interesting to extend the approach that is proposed in this thesis to solve the capacitated examination scheduling problems. We expect that the proposed method can be adapted, in a relatively straightforward manner, to the capacitated scheduling problem by introducing appropriate granular data structures that would permit the required domain transformation in the optimization process.

Apart from this, the investigation of different pre-ordering of exams (for example: other graph coloring heuristics) before real scheduling could be performed to improve the solutions. In addition, further research on how to get the best parameter settings in the search procedure (to be specific the Genetic Algorithm because it was highly dependent on parameter tweaking) in order to guide the searching to obtain global optimum. Having said this, it is also interesting to study when the Genetic Algorithm would outperform the proposed Hill Climbing in the proposed optimization stage.

Lastly, to test the flexibility of the approach, it is recommended that future research attempts to solve other real world examination scheduling problems and randomly generated problems using this approach. Also, other constraints that are suggested in the literature should be taken into consideration in constructing and determining exam schedules.

Bibliography

Abdul-Rahman, S. A., Bargiela, A., Burke, E. K., Ozcan, E., & McCollum, B. (2009, September). Construction of examination timetables based on ordering heuristics. In Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on (pp. 680-685). IEEE.

Abdul-Rahman, S., Burke, E., Bargiela, A., McCollum, B., & Ozcan, E. (2011). A constructive approach to examination timetabling based on adaptive decomposition and ordering (Forthcoming/Available Online).

Abdul-Rahman, S. A., Bargiela, A., Burke, E. K., Ozcan, E., & McCollum, B. & McMullan, P. (2014). Adaptive linear combination of heuristic orderings in constructing examination timetables. European Journal of Operational Research, 232(2), 287-297.

Abdullah, S., Ahmadi, S., Burke, E. K., & Dror, M. (2007). Investigating Ahuja–Orlin's large neighbourhood search approach for examination timetabling. OR Spectrum, 29(2), 351-372.

Abdullah, S., & Alzaqebah, M. (2013). A hybrid self-adaptive bees algorithm for examination timetabling problems. Applied Soft Computing, 13(8), 3608-3620.

Abramson, D., & Abela, J. (1992). A parallel genetic algorithm for solving the school timetabling problem. Division of Information Technology, CSIRO.

Ahmadi, S., Barone, R., Cheng, P., Cowling, P., & McCollum, B. (2003). Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. Proceedings of multidisciplinary international scheduling: theory and applications (MISTA 2003), Nottingham, 155-171.

Al-Betar, M. A., Khader, A. T., & Thomas, J. (2010, August). A combination of metaheuristic components based on harmony search for the uncapacitated examination timetabling. In the 8th Int. Conf. Practice and Theory of Automated Timetabling (PATAT 2010) (pp. 57-80).

Al-Betar, M. A., Khader, A. T., & Doush, I. A. (2014). Memetic techniques for examination timetabling. Annals of Operations Research, 218(1), 23-50.

Al-Yakoob, S. M., Sherali, H. D., & Al-Jazzaf, M. (2010). A mixed-integer mathematical modeling approach to exam timetabling. Computational Management Science, 7(1), 19-46.

Alzaqebah, M., & Abdullah, S. (2014). An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling. Journal of Scheduling, 17(3), 249-262.

Anwar, K., Khader, A. T., Al-Betar, M. A., & Awadallah, M. A. (2013, March). Harmony search-based hyper-heuristic for examination timetabling. In Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on (pp. 176-181). IEEE.

Argile, A., Peytchev, E., Bargiela, A., & Kosonen, I. (1996). DIME: A shared memory environment for distributed simulation, monitoring and control of urban traffic.

Asmuni, H., Burke, E. K., Garibaldi, J. M., & McCollum, B. (2005). Fuzzy multiple heuristic orderings for examination timetabling. In Practice and Theory of Automated Timetabling V (pp. 334-353). Springer Berlin Heidelberg.

Asmuni, H., Burke, E. K., Garibaldi, J. M., & McCollum, B. (2007). A novel fuzzy approach to evaluate the quality of examination timetabling. In Practice and Theory of Automated Timetabling VI (pp. 327-346). Springer Berlin Heidelberg.

Asmuni, H., Burke, E. K., Garibaldi, J. M., McCollum, B., & Parkes, A. J. (2009). An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. Computers & Operations Research, 36(4), 981-1001.

Bardadym, V. A. (1996). Computer-aided school and university timetabling: The new wave. In Practice and theory of automated timetabling (pp. 22-45). Springer Berlin Heidelberg.

Bargiela, A. (1985, November). An algorithm for observability determination in water-system state estimation. In Control Theory and Applications, IEE Proceedings D (Vol. 132, No. 6, pp. 245-250). IET.

Bargiela, A., & Pedrycz, W. (2002). Granular computing: an introduction. Springer.

Bargiela, A., Pedrycz, W., & Hirota, K. (2002). Logic-based granular prototyping. In Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International (pp. 1164-1169). IEEE.

Bargiela, A., Pedrycz, W., & Hirota, K. (2004). Granular prototyping in fuzzy clustering. Fuzzy Systems, IEEE Transactions on, 12(5), 697-709.

Bargiela, A., & Pedrycz, W. (2008). Toward a theory of granular computing for human-centered information processing. Fuzzy Systems, IEEE Transactions on, 16(2), 320-330.

Boizumault, P., Delon, Y., & Peridy, L. (1996). Constraint logic programming for examination timetabling. The Journal of Logic Programming, 26(2), 217-233.

Bolaji, A. L. A., Khader, A. T., Al-Betar, M. A., Awadallah, M. A., & Thomas, J. J. (2012, August). The effect of neighborhood structures on examination timetabling with artificial bee colony. In 9th International Conference on the Practice and Theories of Automated Timetabling (PATAT 2012) (pp. 131-144).

Bosch, R. and Trick, M. (2005). Search Methodologies: Introductory tutorials in optimisation and decision support techniques, chapter Integer Programming, (pp. 69-96). Berlin:Springer, Berlin.

Broder, S. (1964). Final examination scheduling. Communications of the ACM,7(8), 494-498.

Burke, E. K., Elliman, D. G., & Weare, R. F. (1994a). A Genetic Algorithm for University Timetabling, in proceedings of the AISB workshop on Evolutionary Computing. University of Leeds, UK.

Burke, E. K., Elliman, D. G., & Weare, R. F. (1994b, September). A genetic algorithm based university timetabling system. In East-West Conference on Computer Technologies in Education, Crimea, Ukraine pp35-40.

Burke, E. K., Elliman, D. G., & Weare, R. (1994c). A university timetabling system based on graph colouring and constraint manipulation. Journal of Research on Computing in Education, 27, 1-1.

Burke, E. K., Newall, J. P., & Weare, R. F. (1996b). A memetic algorithm for university exam timetabling. In Practice and Theory of Automated Timetabling(pp. 241-250). Springer Berlin Heidelberg.

Burke, E., Jackson, K., Kingston, J. H., & Weare, R. (1997). Automated university timetabling: The state of the art. The computer journal, 40(9), 565-571.

Burke, E. K., Newall, J. P., & Weare, R. F. (1998, August). A simple heuristically guided search for the timetable problem. In Proceedings of the international ICSC symposium on engineering of intelligent systems (EIS98)(pp. 574-579).

Burke, E. K., & Newall, J. P. (1999). A multistage evolutionary algorithm for the timetable problem. Evolutionary Computation, IEEE Transactions on, 3(1), 63-74.

Burke, E., Bykov, Y., & Petrovic, S. (2001). A multicriteria approach to examination timetabling. In Practice and Theory of Automated Timetabling III(pp. 118-131). Springer Berlin Heidelberg.

Burke, E. K., & Petrovic, S. (2002). Recent research directions in automated timetabling. European Journal of Operational Research, 140(2), 266-280.

Burke, E. K., & Newall, J. P. (2004a). Solving examination timetabling problems through adaption of heuristic orderings. Annals of operations Research, 129(1-4), 107-134.

Burke, E., Bykov, Y., Newall, J., & Petrovic, S. (2004b). A time-predefined local search approach to exam timetabling problems. IIE Transactions, 36(6), 509-528.

Burke, E. K., Kingston, J., & De Werra, D. (2004c). 5.6: Applications to Timetabling. Handbook of graph theory, 445.

Burke, E., Dror, M., Petrovic, S., & Qu, R. (2005a). Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems. In The next wave in computing, optimization, and decision technologies (pp. 79-91). Springer US.

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case-based heuristic selection for timetabling problems. Journal of Scheduling, 9(2), 115-132.

Burke, E. K., & Bykov, Y. (2006, August). Solving exam timetabling problems with the flex-deluge algorithm. In Proceedings of PATAT (Vol. 2006, pp. 370-372).

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176(1), 177-192.

Burke, E. K., & Bykov, Y. (2008, August). A late acceptance strategy in hill-climbing for exam timetabling problems. In PATAT 2008 Conference, Montreal, Canada.

Burke, E. K., & Bykov, Y. (2012). The late acceptance hill-climbing heuristic (Vol. 192). technical report CSM.

Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S., & Qu, R. (2010a). Hybrid variable neighbourhood approaches to university exam timetabling.European Journal of Operational Research, 206(1), 46-53.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010b). A classification of hyper-heuristic approaches. In Handbook of Metaheuristics (pp. 449-468). Springer US.

Burke, E. K., Pham, N., Qu, R., & Yellen, J. (2010c). Linear combinations of heuristics for examination timetabling. Annals of Operations Research, 194(1), 89-109.

Burke, E. K., Qu, R., & Soghier, A. (2010e). Adaptive selection of heuristics for improving constructed exam timetables. proceedings of PATAT10, 136-151.

Brélaz, D. (1979). New methods to color the vertices of a graph.Communications of the ACM, 22(4), 251-256.

Caramia, M., Dell'Olmo, P., & Italiano, G. F. (2008). Novel local-search-based approaches to university examination timetabling. INFORMS Journal on Computing, 20(1), 86-99.

Carter, M. (1983). A decomposition algorithm for practical timetabling problems. Department of Industrial Engineering, University of Toronto.

Carter, M. W. (1986). OR Practice—A Survey of Practical Applications of Examination Timetabling Algorithms. Operations research, 34(2), 193-202.

Carter, M. W., Laporte, G., & Chinneck, J. W. (1994). A general examination scheduling system. Interfaces, 24(3), 109-120.

Carter, M. W., Laporte, G., & Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society, 373-383.

Carter, M. W., & Laporte, G. (1996). Recent developments in practical examination timetabling (pp. 1-21). Springer Berlin Heidelberg.

Carter, M. W., & Johnson, D. G. (2001). Extended clique initialisation in examination timetabling. Journal of the Operational Research Society, 538-544.

Carrington, J. R., Pham, N., Qu, R., & Yellen, J. (2007, December). An enhanced weighted graph model for examination/course timetabling. InProceedings of the 26th workshop of the UK planning and scheduling special interest group (pp. 9-16).

Casey, S., & Thompson, J. (2003). GRASPing the examination scheduling problem. In Practice and Theory of Automated Timetabling IV (pp. 232-244). Springer Berlin Heidelberg.

Chen, X., & Bushnell, M. L. (1995). Efficient branch and bound search with application to computer-aided design. Kluwer Academic Publishers.

Cheong, C. Y., Tan, K. C., & Veeravalli, B. (2007, April). Solving the exam timetabling problem via a multi-objective evolutionary algorithm-a more general approach. In Computational Intelligence in Scheduling, 2007. SCIS'07. IEEE Symposium on (pp. 165-172). IEEE.

Cole, A. J. (1964). The preparation of examination time-tables using a small-store computer. The Computer Journal, 7(2), 117-121.

Colijn, A. W., & Layfield, C. (1995). Conflict reduction in examination schedules. In 1995). Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling. 30th Aug-1st Sep (pp. 297-307).

Cooper, T. B., & Kingston, J. H. (1996). The complexity of timetable construction problems (pp. 281-295). Springer Berlin Heidelberg.

Corr, P. H., McCollum, B., McGreevy, M. A. J., & McMullan, P. (2006). A new neural network based construction heuristic for the examination timetabling problem. In Parallel Problem Solving from Nature-PPSN IX (pp. 392-401). Springer Berlin Heidelberg.

Côté, P., Wong, T., & Sabourin, R. (2005). A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In Practice and Theory of Automated Timetabling V (pp. 294-312). Springer Berlin Heidelberg.

David, P. (1998). A constraint-based approach for examination timetabling using local repair techniques. In Practice and Theory of Automated Timetabling II (pp. 169-186). Springer Berlin Heidelberg.

Demeester, P., Bilgin, B., De Causmaecker, P., & Berghe, G. V. (2012). A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. Journal of Scheduling, 15(1), 83-103.

Di Gaspero, L., & Schaerf, A. (2001). Tabu search techniques for examination timetabling. In Practice and Theory of Automated Timetabling III (pp. 104-117). Springer Berlin Heidelberg.

Di Gaspero, L. (2002, August). Recolour, shake and kick: A recipe for the examination timetabling problem. In Proceedings of the fourth international conference on the practice and theory of automated timetabling, Gent, Belgium(pp. 404-407).

Dowsland, K. A., & Thompson, J. M. (2005). Ant colony optimization for the examination scheduling problem. Journal of the Operational Research Society,56(4), 426-438.

Duong, T. A., & Lam, K. H. (2004). Combining Constraint Programming and Simulated Annealing on University Exam Timetabling. InRIVF (pp. 205-210).

Eley, M. (2007). Ant algorithms for the exam timetabling problem. In Practice and Theory of Automated Timetabling VI (pp. 364-382). Springer Berlin Heidelberg.

Erben, W. (2001). A grouping genetic algorithm for graph colouring and exam timetabling. In Practice and Theory of Automated Timetabling III (pp. 132-156). Springer Berlin Heidelberg.

Ersoy, E., Özcan, E., & Uyar, Ş. (2007, August). Memetic algorithms and hyperhill-climbers. In Proc. of the 3rd Multidisciplinary Int. conf. on scheduling: theory and applications, P. Baptiste, G. Kendall, AM Kordon and F. Sourd, Eds(pp. 159-166).

Even, S., Itai, A., & Shamir, A. (1976). On the complexity of time table and multi-commodity flow problems. In Foundations of Computer Science, 1975., 16th Annual Symposium on (pp. 184-193). IEEE.

Gogos, C., Alefragis, P., & Housos, E. (2012). An improved multi-staged algorithmic process for the solution of the examination timetabling problem.Annals of Operations Research, 194(1), 203-221.

Guéret, C., Jussien, N., Boizumault, P., & Prins, C. (1996). Building university timetables using constraint logic programming. In Practice and Theory of Automated Timetabling (pp. 130-145). Springer Berlin Heidelberg.

Gunawan, A., Ng, K. M., & Poh, K. L. (2007a). An improvement heuristic for the timetabling problem. International Journal of Computational Science, 1(2), 162-178.

Gunawan, A., Ng, K. M., & Poh, K. L. (2007b). Solving the teacher assignment-course scheduling problem by a hybrid algorithm. World Academy of Science, Engineering and Technology, 33, 259-264.

Gunawan, A., Ng, K. M., & Poh, K. L. (2008, August). A hybrid algorithm for the university course timetabling problem. In Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling.

Gyori, S., Petres, Z., & Várkonyi-Kóczy, A. R. (2001). Genetic Algorithms in Timetabling. A New Approach. Budapest University of Technology and Economics, Hungary.

Hertz, A. (1991). Tabu search for large scale timetabling problems. European journal of operational research, 54(1), 39-47.

Johnson, D. (1990). Timetabling university examinations. Journal of the Operational Research Society, 39-47.

Joslin, D. E., & Clements, D. P. (1999, July). "Squeaky Wheel" Optimization. In AAAI/IAAI (pp. 340-346).

Kahar, M. N. M., & Kendall, G. (2010). The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution. European journal of operational research, 207(2), 557-565.

Kendall, G., & Hussin, N. M. (2005a). An investigation of a tabu-search-based hyper-heuristic for examination timetabling.

In Multidisciplinary Scheduling: Theory and Applications (pp. 309-328). Springer US.

Kendall, G., & Hussin, N. M. (2005b). A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. InPractice and Theory of Automated Timetabling V (pp. 270-293). Springer Berlin Heidelberg.

Kendall, G., & Li, J. (2007). Combining examinations to accelerate timetable construction. In Proceedings of The 7th International Conference on the Practice and Theory of Automated Timetabling, Montreal.

Kiaer, L., & Yellen, J. (1992). Weighted graphs and university course timetabling. Computers & operations research, 19(1), 59-67.

Kristiansen, S., & Stidsen, T. R. (2013). A comprehensive study of educational timetabling-a survey. Department of Management Engineering, Technical University of Denmark.

Laporte, G., & Desroches, S. (1984). Examination timetabling by computer.Computers & Operations Research, 11(4), 351-360.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. OR spectrum, 30(1), 167-190.

McCollum, B. (2007). A perspective on bridging the gap between theory and practice in university timetabling. In Practice and theory of automated timetabling VI (pp. 3-23). Springer Berlin Heidelberg.

McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K., & Qu, R. (2012). A new model for automated examination timetabling. Annals of Operations Research, 194(1), 291-315.

Mehta, N. K. (1981). The application of a graph coloring method to an examination scheduling problem. Interfaces, 11(5), 57-65.

Merlot, L. T., Boland, N., Hughes, B. D., & Stuckey, P. J. (2003). A hybrid algorithm for the examination timetabling problem. In Practice and theory of automated timetabling IV (pp. 207-231). Springer Berlin Heidelberg.

MirHassani, S. A. (2006). Improving paper spread in examination timetables using integer programming. Applied mathematics and computation, 179(2), 702-706.

Ozcan, E., & Ersoy, E. (2005, September). Final exam scheduler-FES. InEvolutionary Computation, 2005. The 2005 IEEE Congress on (Vol. 2, pp. 1356-1363). IEEE.

Ozcan, E., Bykov, Y., Birben, M., & Burke, E. K. (2009, May). Examination timetabling using late acceptance hyper-heuristics. In Evolutionary Computation, 2009. CEC'09. IEEE Congress on (pp. 997-1004). IEEE.

Pais, T. C., & Burke, E. (2010). Choquet integral for combining heuristic values for exam timetabling problem. PATAT 2010, 305.

Paquete, L. F., & Fonseca, C. M. (2001, March). A study of examination timetabling with multiobjective evolutionary algorithms. In Proceedings of the 4th Metaheuristics International Conference (MIC 2001) (pp. 149-154).

Peck, J. E. L., & Williams, M. R. (1966). Algorithm 286: Examination scheduling. Communications of the ACM, 9(6), 433-434.

Pedrycz, W., Smith, M. H., & Bargiela, A. (2000). A granular signature of data. In Fuzzy Information Processing Society, 2000. NAFIPS. 19th International Conference of the North American (pp. 69-73). IEEE.

Petrovic, S., & Bykov, Y. (2003). A multiobjective optimisation technique for exam timetabling based on trajectories. In Practice and Theory of Automated Timetabling IV (pp. 181-194). Springer Berlin Heidelberg.

Petrovic, S., & Burke, E. K. (2004). University Timetabling. Ch. 45 in the Handbook of Scheduling: Algorithms, Models, and Performance Analysis (eds. J. Leung), Chapman Hall.

Petrovic, S., Yang, Y., & Dror, M. (2005). Case-based initialisation of metaheuristics for examination timetabling. In Multidisciplinary scheduling: theory and applications (pp. 289-308). Springer US.

Peytchev, E. T., Bargiela, A., & Gessing, R. (1996, October). A predictive macroscopic city traffic flows simulation model. In 8th European Simulation Symposium, Genoa, Italy, ISBN (pp. 1-565555).

Pillay, N., & Banzhaf, W. (2009). A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem.European Journal of Operational Research, 197(2), 482-491.

Pillay, N. (2013). A survey of school timetabling research. Annals of Operations Research, 218(1), 261-293.

Qu, R., & Burke, E. (2005). Hybrid variable neighborhood hyperheuristics for exam timetabling problems.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., & Lee, S. Y. (2006). A survey of search methodologies and automated approaches for examination timetabling. Computer Science Technical Report No. NOTTCS-TR-2006-4, UK.

Qu, R., & Burke, E. K. (2007). Adaptive decomposition and construction for examination timetabling problems. Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications, 418-425.

Qu, R., & Burke, E. K. (2009). Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. Journal of the Operational Research Society, 60(9), 1273-1285.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., & Lee, S. Y. (2009a). A survey of search methodologies and automated system development for examination timetabling. Journal of scheduling, 12(1), 55-89.

Qu, R., Burke, E. K., & McCollum, B. (2009b). Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems.European Journal of Operational Research, 198(2), 392-404.

Qu, R., He, F., & Burke, E. K. (2009c). Hybridizing integer programming models with an adaptive decomposition approach for exam timetabling problems. The 4th Multidisciplinary International Scheduling: Theory and Applications, 435-446.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2009). Granular Modelling Of Exam To Slot Allocation. In ECMS (pp. 861-866).

Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2012). Domain transformation approach to deterministic optimization of examination timetables. Artificial Intelligence Research, 2(1), p122.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2013a). Hill climbing versus genetic algorithm optimization in solving the examination timetabling problem.

Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2013b). Analysis Of Backtracking In University Examination Scheduling. In ECMS (pp. 782-787).

Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2013c). A Study on the Effectiveness of Genetic Algorithm and Identifying the Best Parameters Range for Slots Swapping in the Examination Scheduling.

Reis, L. P., & Oliveira, E. (1999). Constraint logic programming using set variables for solving timetabling problems. In 12th international conference on applications of Prolog.

Ross, P., & Corne, D. (1995). Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems. In Evolutionary Computing (pp. 94-102). Springer Berlin Heidelberg.

Ross, P., Hart, E., & Corne, D. (1998). Some observations about GA-based exam timetabling. In Practice and Theory of Automated Timetabling II (pp. 115-129). Springer Berlin Heidelberg.

Sabar, N. R., Ayob, M., & Kendall, G. (2009, August). Solving examination timetabling problems using honey-bee mating optimization (ETP-HBMO). InProceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), Dublin, Ireland (pp. 399-408).

Sabar, N. R., Ayob, M., & Kendall, G. (2009, April). Tabu exponential Monte-Carlo with counter heuristic for examination timetabling. In Computational Intelligence in Scheduling, 2009. CI-Sched'09. IEEE Symposium on (pp. 90-94). IEEE.

Sabar, N. R., Ayob, M., Qu, R., & Kendall, G. (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence, 37(1), 1-11.

Schaerf, A. (1999). A survey of automated timetabling. Artificial intelligence review, 13(2), 87-127.

Schmidt, G., & Ströhlein, T. (1980). Timetable construction–an annotated bibliography. The Computer Journal, 23(4), 307-316.

Sierksma, G. (2001). Linear and integer programming: theory and practice. CRC Press.

Terashima-Marín, H., Ross, P., & Valenzuela-Rendón, M. (1999). Application of the hardness theory when solving the timetabling problem with genetic algorithms. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on (Vol. 1). IEEE.

Thomas, J. J., Khader, A. T., & Belaton, B. (2009, August). Visualization Techniques on the Examination Timetabling Pre-processing Data. In Computer Graphics, Imaging and Visualization, 2009. CGIV'09. Sixth International Conference on (pp. 454-458). IEEE.

Thompson, J. M., & Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. Computers & Operations Research,25(7), 637-648.

Turabieh, H., & Abdullah, S. (2011a). An integrated hybrid approach to the examination timetabling problem. Omega, 39(6), 598-607.

Turabieh, H., & Abdullah, S. (2011b). A hybrid fish swarm optimisation algorithm for solving examination timetabling problems. In Learning and Intelligent Optimization (pp. 539-551). Springer Berlin Heidelberg.

Ülker, Ö., Özcan, E., & Korkmaz, E. E. (2007). Linear linkage encoding in grouping problems: applications on graph coloring and timetabling.

In Practice and Theory of Automated Timetabling VI (pp. 347-363). Springer Berlin Heidelberg.

Weare, R., Burke, E., & Elliman, D. (1995, January). A hybrid genetic algorithm for highly constrained timetabling problems. In Proceedings of the Sixth International Conference on Genetic Algorithms, ed. LJ Eshelman (pp. 605-610).

Welsh, D. J., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal,10(1), 85-86.

White, G. M., & Xie, B. S. (2001). Examination timetables and tabu search with longer-term memory. In Practice and Theory of Automated Timetabling III (pp. 85-103). Springer Berlin Heidelberg.

White, G. M., Xie, B. S., & Zonjic, S. (2004). Using tabu search with longer-term memory and relaxation to create examination timetables. European Journal of Operational Research, 153(1), 80-91.

Wong, T., Bigras, P., & de Kelper, B. (2005, October). A multi-neighborhood and multi-operator strategy for the uncapacitated exam proximity problem. InSystems, Man and Cybernetics, 2005 IEEE International Conference on (Vol. 4, pp. 3810-3816). IEEE.

Wilson R. (2010). Ron Wilson Design, [Online] Available at http://www.acrolite.org/

Wood, D. C. (1968). A system for computing university examination timetables.The Computer Journal, 11(1), 41-47.

Wren, A. (1996). Scheduling, timetabling and rostering—a special relationship?. In Practice and theory of automated timetabling (pp. 46-75). Springer Berlin Heidelberg.

Yang, Y., & Petrovic, S. (2004). A novel similarity measure for heuristic selection in examination timetabling. In Practice and Theory of Automated Timetabling V (pp. 247-269). Springer Berlin Heidelberg.